Amazon Mechanical Turk

API Reference

API Version 2017-01-17

Amazon Mechanical Turk: API Reference

Table of Contents

Amazon Mechanical Turk API Reference

This is the *Amazon Mechanical Turk API Reference*. This guide provides detailed information about Amazon Mechanical Turk operations, data structures, and parameters. The major sections of this guide are described in the following table.

Amazon Mechanical Turk is a web service that provides an on-demand, scalable, human workforce to complete jobs that humans can do better than computers, for example, recognizing objects in photos. For more information about this product go to the Amazon Mechanical Turk website.

Operations (p. 2)	Alphabetical list of all Amazon Mechanical Turk operations.
Data Structure Schema Locations (p. 122)	Links to Amazon Mechanical Turk data structure schemas.
Question and Answer Data (p. 81)	Description of question and answer data that Amazon Mechanical Turk passes between Requesters and Workers.
Data Structures (p. 124)	Alphabetical list of all Amazon Mechanical Turk data structures.
Review Policies (p. 155)	Description of Amazon Mechanical Turk Review Policies.
Managing Notifications (p. 167)	Description of how Amazon Mechanical Turk sends notification messages to your application.

Operations

The Amazon Mechanical Turk API consists of web service operations for every task the service can perform. This section describes each operation in detail.

AcceptQualificationRequest

Description

The `AcceptQualificationRequest` operation grants a Worker's request for a Qualification.

Only the owner of the Qualification type can grant a Qualification request for that type.

Request Syntax

```
{
  "QualificationRequestId": String,

  "IntegerValue": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationRequestId`	The ID of the Qualification request, as returned by the ListQualificationRequests (p. 52) operation. Type: String.	Yes
`IntegerValue`	The value of the Qualification. You can omit this value if you are using the presence or absence of the Qualification as the basis for a HIT requirement. Type: Integer Default: 1	No

Response Elements

A successful request for the `AcceptQualificationRequest` operation returns with no errors and an empty body.

Example

The following example shows how to use the `AcceptQualificationRequest` operation:

Sample Request

The following example grants a Qualification to a user.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
```

```
X-Amz-Date: <Date>
{
  QualificationRequestId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  IntegerValue:95
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

ApproveAssignment

Description

The `ApproveAssignment` operation approves the results of a completed assignment created with the API.

Approving an assignment initiates two payments from the Requester's Amazon.com account: the Worker who submitted the results is paid the reward specified in the HIT, and Amazon Mechanical Turk fees are debited. If the Requester's account does not have adequate funds for these payments, the call to ApproveAssignment returns an exception, and the approval is not processed. You can include an optional feedback message with the approval, which the Worker can see in the Status section of the web site.

You can also call this operation on assignments that were previous rejected and approve them by over-riding the previous rejection. This works only on rejected assignments that were submitted within the previous 30 days and only if the assignment's related HIT has not been deleted.

Note
To maintain the consistency of the assignments for HITs in a batch, if a HIT is created through the Amazon Mechanical Turk Requester website, you must use the Requester website UI to approve or reject the work.
If you want to be able to approve or reject work through the API, you can use a HITLayout (p. 86) when calling CreateHIT (p. 12) to utilize an existing template from the Amazon Mechanical Turk Requester website

Request Syntax

```
{
  "AssignmentId": String,

  "RequesterFeedback": String,

  "OverrideRejection": Boolean
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`AssignmentId`	The ID of the assignment. This parameter must correspond to a HIT created by the Requester. Type: String	Yes
`RequesterFeedback`	A message for the Worker, which the Worker can see in the Status section of the web site. Type: String Constraints: Can be up to 1024 characters (including multi-byte characters). The RequesterFeedback parameter cannot contain ASCII characters 0-8, 11,12, or 14-31. If these characters are present, the operation throws an InvalidParameterValue error.	No

Name	Description	Required
`OverrideRejection`	A flag indicating whether you want to approve an assignment that was previously rejected. Type: Boolean Constraints: This works only on rejected assignments that were submitted within the previous 30 days and only if the assignment's related HIT has not been deleted.	No

Response Elements

A successful request for the `ApproveAssignment` operation returns with no errors and an empty body.

Example

The following example shows how to use the `ApproveAssignment` operation:

Sample Request

The following example approves an assignment identified by its assignment ID.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  AssignmentId:"123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

AssociateQualificationWithWorker

Description

The `AssociateQualificationWithWorker` operation gives a Worker a Qualification. `AssociateQualificationWithWorker` does not require that the Worker submit a Qualification request. It immediately gives the Worker the Qualification.

You can only assign a Qualification of a Qualification type that you created (using the CreateQualificationType (p. 24) operation).

> **Tip**
> `AssociateQualificationWithWorker` does not affect any pending Qualification requests for the Qualification by the Worker. If you associate a Qualification to a Worker, then later accept a Qualification request made by the Worker, accepting the request may modify the Qualification score. To resolve a pending Qualification request without affecting the Qualification the Worker already has, reject the request with the RejectQualificationRequest (p. 67) operation.

Request Syntax

```
{
  "QualificationTypeId": String,

  "WorkerId": String,

  "IntegerValue": Integer,

  "SendNotification": Boolean
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the Qualification type to use for the assigned Qualification. Type: String Constraints: must be a valid Qualification type ID, as returned by the CreateQualificationType (p. 24) operation.	Yes
`WorkerId`	The ID of the Worker to whom the Qualification is being assigned. Worker IDs are included with submitted HIT assignments and Qualification requests. Type: String	Yes
`IntegerValue`	The value of the Qualification to assign. Type: Integer	Yes

Name	Description	Required
	Default: 1	
SendNotification	Specifies whether to send a notification email message to the Worker saying that the qualification was assigned to the Worker. Type: Boolean Default: true	No

Response Elements

A successful request for the AssociateQualificationWithWorker operation returns with no errors and an empty body.

Example

The following example shows how to use the AssociateQualificationWithWorker operation:

Sample Request

The following example grants a Qualification to a Worker.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  WorkerId:"AZ3456EXAMPLE",
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  IntegerValue:1,
  SendNotification:false
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

CreateAdditionalAssignmentsForHIT

Description

The `CreateAdditionalAssignmentsForHIT` operation increases the maximum number of assignments of an existing HIT.

To extend the maximum number of assignments, specify the number of additional assignments.

Note

- HITs created with fewer than 10 assignments cannot be extended to have 10 or more assignments. Attempting to add assignments in a way that brings the total number of assignments for a HIT from fewer than 10 assignments to 10 or more assignments will result in an `AWS.MechanicalTurk.InvalidMaximumAssignmentsIncrease` exception.
- HITs that were created before July 22, 2015 cannot be extended. Attempting to extend HITs that were created before July 22, 2015 will result in an `AWS.MechanicalTurk.HITTooOldForExtension` exception.

Request Syntax

```
{
  "HITId": String,

  "NumberOfAdditionalAssignments": Integer,

  "UniqueRequestToken": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The ID of the HIT to for which to request more assignments. Type: String	Yes
`NumberOfAdditionalAssignments`	The number of additional assignments to request for this HIT. Type: Integer	Yes
`UniqueRequestToken`	A unique identifier for this request, which allows you to retry the call on error without extending the HIT multiple times. This is useful in cases such as network timeouts where it is unclear whether or not the call succeeded on the server. If the extend HIT already exists in the system from a previous call using the same `UniqueRequestToken`, subsequent calls will return an error with a message containing the request ID.	No

Name	Description	Required
	Type: String Constraints: must not be longer than 64 characters in length.	

Response Elements

A successful request for the `CreateAdditionalAssignmentsForHIT` operation returns with no errors and an empty body.

Example

The following example shows how to use the `CreateAdditionalAssignmentsForHIT` operation:

Sample Request

The following example adds 2 more assignments to a HIT.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  HITId:"123RVAZW00EXAMPLE2SL3LSK",
  HITNumberOfAdditionalAssignments:2
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

CreateHIT

Description

The `CreateHIT` operation creates a new HIT (p. 128) (Human Intelligence Task). The new HIT is made available for Workers to find and accept on the Amazon Mechanical Turk website.

This operation allows you to specify a new HIT by passing in values for the properties of the HIT, such as its title, reward amount and number of assignments. When you pass these values to `CreateHIT`, a new HIT is created for you, with a new `HITTypeID`.

CreateHIT also supports several ways to provide question data: by providing a value for the Question (p. 81) parameter that fully specifies the contents of the HIT, or by providing a HitLayoutId (p. 86) and associated `HitLayoutParameters`.

For a step-by-step tutorial, please read our blog post which walks you through the HIT creation process.

Note

- If a HIT is created with 10 or more maximum assignments, there is an additional fee. For more information, see Amazon Mechanical Turk Pricing.
- During the first few seconds after calling `CreateHIT`, the HIT cannot be modified except to expire it using `UpdateExpirationForHIT`. Until the HIT becomes modifiable, the following operations may return a RequestError:

  `UpdateHITTypeOfHIT`, `CreateAdditionalAssignmentsForHIT`, and `UpdateExpirationForHIT` with a DateTime in the future.

Request Syntax

```
{
  "Title": String,

  "Description": String,

  "Question": String,

  "HITLayoutId": String,

  "HITLayoutParameters": HITLayoutParameterList (p. 133),

  "Reward": String,

  "AssignmentDurationInSeconds": Integer,

  "LifetimeInSeconds": Integer,

  "Keywords": String,

  "MaxAssignments": Integer,

  "AutoApprovalDelayInSeconds": Integer,

  "QualificationRequirements": QualificationRequirementList (p. 138),

  "AssignmentReviewPolicy": ReviewPolicy (p. 150),

  "HITReviewPolicy": ReviewPolicy (p. 150),
```

```
 "RequesterAnnotation": String,

 "UniqueRequestToken": String
}
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Title`	The title of the HIT. A title should be short and descriptive about the kind of task the HIT contains. On the Amazon Mechanical Turk web site, the HIT title appears in search results, and everywhere the HIT is mentioned. Type: String	Yes
`Description`	A general description of the HIT. A description includes detailed information about the kind of task the HIT contains. On the Amazon Mechanical Turk web site, the HIT description appears in the expanded view of search results, and in the HIT and assignment screens. A good description gives the user enough information to evaluate the HIT before accepting it. Type: String	Yes
`Question`	The data the person completing the HIT uses to produce the results. You can learn more about the various ways of specifying this field here (p. 81) Type: String Constraints: The XML question data must not be larger than 64 kilobytes (65,535 bytes) in size, including whitespace. Either a Question parameter or a HITLayoutId parameter must be provided.	No
`HITLayoutId`	The HITLayoutId allows you to use a pre-existing HIT design with placeholder values and create an additional HIT by providing those values as HITLayoutParameters. For more information, see here (p. 86). Type: String Constraints: Either a Question parameter or a HITLayoutId parameter must be provided.	No
`HITLayoutParameters`	If the HITLayoutId is provided, any placeholder values must be filled in with values using the HITLayoutParameter structure. For more information, see HITLayout. Type: HITLayoutParameterList (p. 133)	No

Name	Description	Required
`Reward`	The US Dollar amount the Requester will pay a Worker for successfully completing the HIT. Type: String	Yes
`AssignmentDurationInSeconds`	The amount of time, in seconds, that a Worker has to complete the HIT after accepting it. If a Worker does not complete the assignment within the specified duration, the assignment is considered abandoned. If the HIT is still active (that is, its lifetime has not elapsed), the assignment becomes available for other users to find and accept. Type: Integer	Yes
`LifetimeInSeconds`	An amount of time, in seconds, after which the HIT is no longer available for users to accept. After the lifetime of the HIT elapses, the HIT no longer appears in HIT searches, even if not all of the assignments for the HIT have been accepted. Type: Integer	Yes
`Keywords`	One or more words or phrases that describe the HIT, separated by commas. These words are used in searches to find HITs. Type: String	Yes
`MaxAssignments`	The number of times the HIT can be accepted and completed before the HIT becomes unavailable. Type: Integer	Yes
`AutoApprovalDelayInSeconds`	The number of seconds after an assignment for the HIT has been submitted, after which the assignment is considered Approved automatically unless the Requester explicitly rejects it. Type: Integer	No
`QualificationRequirements`	A condition that a Worker's Qualifications must meet before the Worker is allowed to accept and complete the HIT. Type: QualificationRequirementList (p. 146)	No
`AssignmentReviewPolicy`	The Assignment-level Review Policy applies to the assignments under the HIT. You can specify for Mechanical Turk to take various actions based on the policy. Type: ReviewPolicy (p. 150)	No

Name	Description	Required
`HITReviewPolicy`	The HIT-level Review Policy applies to the HIT. You can specify for Mechanical Turk to take various actions based on the policy. Type: ReviewPolicy (p. 150)	No
`RequesterAnnotation`	An arbitrary data field. The RequesterAnnotation parameter lets your application attach arbitrary data to the HIT for tracking purposes. For example, this parameter could be an identifier internal to the Requester's application that corresponds with the HIT. The RequesterAnnotation parameter for a HIT is only visible to the Requester who created the HIT. It is not shown to the Worker, or any other Requester. The RequesterAnnotation parameter may be different for each HIT you submit. It does not affect how your HITs are grouped. Type: String Constraints: must not be longer than 255 characters in length.	No
`UniqueRequestToken`	A unique identifier for this request. Allows you to retry the call on error without creating duplicate HITs. This is useful in cases such as network timeouts where it is unclear whether or not the call succeeded on the server. If the HIT already exists in the system from a previous call using the same UniqueRequestToken, subsequent calls will return a AWS.MechanicalTurk.HitAlreadyExists error with a message containing the HITId. Type: String Constraints: must not be longer than 64 characters in length. It is your responsibility to ensure uniqueness of the token. The unique token expires after 24 hours. Subsequent calls using the same UniqueRequestToken made after the 24 hour limit could create duplicate HITs.	No

Response Elements

A successful request for the CreateHIT operation returns HIT (p. 128) object containing the newly created HIT data.

Example

The following example shows how to use the `CreateHIT` operation:

Sample Request

The following example creates a simple HIT. The Question parameter takes a block of XML data as its value.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  Title:"Compare two photographs",
  Description:"Compare two pictures and pick one",
  Reward:0.5,
  Question:[XML question data] (p. 81),
  AssignmentDurationInSeconds:0,
  LifetimeInSeconds:604800,
  Keywords:"location, photograph, image, identification, opinion"
}
```

Tip
Find code samples for MTurk Requester API at AWSLabs on Github

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  HITId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  HITTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE"
}
```

CreateHITType

Description

The `CreateHITType` operation creates a new HIT type.

`CreateHITType` lets you be explicit about which HITs ought to be the same type. It also gives you error checking, to ensure that you call the CreateHITWithHITType (p. 20) operation with a valid HIT type ID.

If you register a HIT type with values that match an existing HIT type, the HIT type ID of the existing type will be returned.

Request Syntax

```
{
  "Title": String,

  "Description": String,

  "Reward": String,

  "AssignmentDurationInSeconds": Integer,

  "Keywords": String,

  "AutoApprovalDelayInSeconds": Integer,

  "QualificationRequirements": QualificationRequirementList (p. 138)
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Title`	The title of the HIT. A title should be short and descriptive about the kind of task the HIT contains. On the Amazon Mechanical Turk web site, the HIT title appears in search results, and everywhere the HIT is mentioned. Type: String	Yes
`Description`	A general description of the HIT. A description includes detailed information about the kind of task the HIT contains. On the Amazon Mechanical Turk web site, the HIT description appears in the expanded view of search results, and in the HIT and assignment screens. A good description gives the user enough information to evaluate the HIT before accepting it. Type: String	Yes
`Reward`	The US Dollar amount the Requester will pay a Worker for successfully completing the HIT.	Yes

Name	Description	Required
	Type: String	
AssignmentDurationInSeconds	The amount of time, in seconds, that a Worker has to complete the HIT after accepting it. If a Worker does not complete the assignment within the specified duration, the assignment is considered abandoned. If the HIT is still active (that is, its lifetime has not elapsed), the assignment becomes available for other users to find and accept. Type: Integer	Yes
Keywords	One or more words or phrases that describe the HIT, separated by commas. These words are used in searches to find HITs. Type: String	Yes
AutoApprovalDelayInSeconds	The number of seconds after an assignment for the HIT has been submitted, after which the assignment is considered Approved automatically unless the Requester explicitly rejects it. Type: Integer	No
QualificationRequirements	A condition that a Worker's Qualifications must meet before the Worker is allowed to accept and complete the HIT. Type: QualificationRequirementList (p. 138)	No

Response Elements

A successful request for the CreateHITWithHITType operation returns a HITTypeId. A HITTypeId can be up to 255 bytes long.

Example

The following example shows how to use the CreateHITType operation:

Sample Request

The following example creates a new HIT type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  Title:"Compare two photographs",
  Description:"Compare two pictures and pick one",
  Reward:0.5,
  AssignmentDurationInSeconds:0,
  Keywords:"location, photograph, image, identification, opinion"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  HITTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE"
}
```

CreateHITWithHITType

Description

The `CreateHITWithHITType` operation creates a new Human Intelligence Task (HIT) using an existing HITTypeID generated by the `CreateHITType` operation.

This is an alternative way to create HITs from the `CreateHIT` operation. This is the recommended best practice for Requesters who are creating large numbers of HITs.

CreateHIT also supports several ways to provide question data: by providing a value for the Question (p. 81) parameter that fully specifies the contents of the HIT, or by providing a HitLayoutId (p. 86) and associated `HitLayoutParameters`.

> **Note**
> If a HIT is created with 10 or more maximum assignments, there is an additional fee. For more information, see Amazon Mechanical Turk Pricing.

Request Syntax

```
{
  "HITTypeId": String,

  "Question": String,

  "HITLayoutId": String,

  "HITLayoutParameters": HITLayoutParameterList (p. 133),

  "LifetimeInSeconds": Integer,

  "MaxAssignments": Integer,

  "AssignmentReviewPolicy": ReviewPolicy (p. 150),

  "HITReviewPolicy": ReviewPolicy (p. 150),

  "RequesterAnnotation": String,

  "UniqueRequestToken": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITTypeId`	The HIT type ID. Type: String	Yes
`Question`	The data the person completing the HIT uses to produce the results. You can learn more about the various ways of specifying this field here (p. 81). Type: String	No

Name	Description	Required
	Constraints: The XML question data must not be larger than 64 kilobytes (65,535 bytes) in size, including whitespace. Either a Question parameter or a HITLayoutId parameter must be provided.	
`HITLayoutId`	The HITLayoutId allows you to use a pre-existing HIT design with placeholder values and create an additional HIT by providing those values as HITLayoutParameters. For more information, see here (p. 86). Type: String Constraints: Either a Question parameter or a HITLayoutId parameter must be provided.	No
`HITLayoutParameters`	If the HITLayoutId is provided, any placeholder values must be filled in with values using the HITLayoutParameter structure. For more information, see HITLayout. Type: HITLayoutParameterList (p. 133)	No
`LifetimeInSeconds`	An amount of time, in seconds, after which the HIT is no longer available for users to accept. After the lifetime of the HIT elapses, the HIT no longer appears in HIT searches, even if not all of the assignments for the HIT have been accepted. Type: Integer	Yes
`MaxAssignments`	The number of times the HIT can be accepted and completed before the HIT becomes unavailable. Type: Integer	Yes
`AssignmentReviewPolicy`	The Assignment-level Review Policy applies to the assignments under the HIT. You can specify for Mechanical Turk to take various actions based on the policy. Type: ReviewPolicy (p. 150)	No
`HITReviewPolicy`	The HIT-level Review Policy applies to the HIT. You can specify for Mechanical Turk to take various actions based on the policy. Type: ReviewPolicy (p. 150)	No

Name	Description	Required
RequesterAnnotation	An arbitrary data field. The RequesterAnnotation parameter lets your application attach arbitrary data to the HIT for tracking purposes. For example, this parameter could be an identifier internal to the Requester's application that corresponds with the HIT. The RequesterAnnotation parameter for a HIT is only visible to the Requester who created the HIT. It is not shown to the Worker, or any other Requester. The RequesterAnnotation parameter may be different for each HIT you submit. It does not affect how your HITs are grouped. Type: String	No
UniqueRequestToken	A unique identifier for this request. Allows you to retry the call on error without creating duplicate HITs. This is useful in cases such as network timeouts where it is unclear whether or not the call succeeded on the server. If the HIT already exists in the system from a previous call using the same UniqueRequestToken, subsequent calls will return a AWS.MechanicalTurk.HitAlreadyExists error with a message containing the HITId. Type: String Constraints: must not be longer than 64 characters in length. It is your responsibility to ensure uniqueness of the token. The unique token expires after 24 hours. Subsequent calls using the same UniqueRequestToken made after the 24 hour limit could create duplicate HITs.	No

Response Elements

A successful request for the CreateHIT operation returns HIT (p. 128) object containing the newly created HIT data.

Example

The following example shows how to use the `CreateHITWithHITType` operation:

Sample Request

The following example creates a simple HIT. The Question parameter takes a block of XML data as its value.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  HITTypeId:"T100CN9P324W00EXAMPLE",
  Question:[XML question data],
```

```
  LifetimeInSeconds:604800
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  HITId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  HITTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE"
}
```

CreateQualificationType

Description

The `CreateQualificationType` operation creates a new Qualification type, which is represented by a QualificationType (p. 146) data structure.

Request Syntax

```
{
  "Name": String,

  "Description": String,

  "Keywords": String,

  "RetryDelayInSeconds": Non-negative integer,

  "QualificationTypeStatus": String,

  "Test": String,

  "AnswerKey": String,

  "TestDurationInSeconds": Integer,

  "AutoGranted": Boolean,

  "AutoGrantedValue": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Name`	The name you give to the Qualification type. The type name is used to represent the Qualification to Workers, and to find the type using a Qualification type search. It must be unique across all of your Qualification types. Type: String	Yes
`Description`	A long description for the Qualification type. On the Amazon Mechanical Turk website, the long description is displayed when a Worker examines a Qualification type. Type: String Constraints: Must be less than or equal to 2000 characters	Yes
`Keywords`	One or more words or phrases that describe the Qualification type, separated by commas. The	No

Name	Description	Required
	keywords of a type make the type easier to find during a search. Type: String Constraints: Must be less than or equal to 1000 characters, including commas and spaces.	
`RetryDelayInSeconds`	The number of seconds that a Worker must wait after requesting a Qualification of the Qualification type before the worker can retry the Qualification request. Type: Non-negative integer Default: None. If not specified, retries are disabled and Workers can request a Qualification of this type only once, even if the Worker has not been granted the Qualification. It is not possible to disable retries for a Qualification type after it has been created with retries enabled. If you want to disable retries, you must delete existing retry-enabled Qualification type and then create a new Qualification type with retries disabled.	No
`QualificationTypeStatus`	The initial status of the Qualification type. Type: String Constraints: Valid values are: Active | Inactive	Yes
`Test`	The questions for the Qualification test a Worker must answer correctly to obtain a Qualification of this type. If this parameter is specified, `TestDurationInSeconds` must also be specified. Type: String Constraints: Must not be longer than 65535 bytes. Must be a QuestionForm data structure. This parameter cannot be specified if AutoGranted is true. Default: None. If not specified, the Worker may request the Qualification without answering any questions.	No
`AnswerKey`	The answers to the Qualification test specified in the Test parameter, in the form of an AnswerKey data structure. Type: String Constraints: Must not be longer than 65535 bytes. Default: None. If not specified, you must process Qualification requests manually.	No

Name	Description	Required
TestDurationInSeconds	The number of seconds the Worker has to complete the Qualification test, starting from the time the Worker requests the Qualification. Type: Integer	Conditional: required when a Test is specified.
AutoGranted	Specifies whether requests for the Qualification type are granted immediately, without prompting the Worker with a Qualification test. Type: Boolean Constraints: If the Test parameter is specified, this parameter cannot be true. Default: False	No
AutoGrantedValue	The Qualification value to use for automatically granted Qualifications. This parameter is used only if the AutoGranted parameter is true. Type: Integer Default: 1 when used with AutoGranted. None when AutoGranted is not specified.	No

Response Elements

A successful request for the CreateQualificationType operation returns a QualificationType (p. 146) data structure.

Example

The following example shows how to use the `CreateQualificationType` operation:

Sample Request

The following example creates a Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  Name:"EnglishWritingAbility",
  Description:"The ability to write and edit in text in English",
  QualificationTypeStatus:"Active"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
```

```
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  Name:"EnglishWritingAbility",
  Description:"The ability to write and edit in text in English",
  QualificationTypeStatus:"Active"
}
```

CreateWorkerBlock

Description

The `CreateWorkerBlock` operation allows you to prevent a Worker from working on your HITs. For example, you can block a Worker who is producing poor quality work. You can block up to 100,000 Workers.

Request Syntax

```
{
  "WorkerId": String,

  "Reason": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`WorkerId`	The ID of the Worker to block Type: String	Yes
`Reason`	A message that explains the reason for blocking the Worker. The Worker does not see this message. Type: String	No

Response Elements

A successful request for the `CreateWorkerBlock` operation returns with no errors and an empty body.

DeleteHIT

Description

The `DeleteHIT` operation disposes of a HIT that is no longer needed. Only the Requester who created the HIT can delete it.

You can only dispose of HITs that are in the Reviewable state, with all of their submitted assignments already either approved or rejected. If you call the DeleteHIT operation on a HIT that is not in the Reviewable state (for example, that has not expired, or still has active assignments), or on a HIT that is Reviewable but without all of its submitted assignments already approved or rejected, the service returns an error.

Note

- HITs are automatically disposed of after 120 days.
- After you dispose of a HIT, you can no longer approve the HIT's rejected assignments.
- Disposed of HITs are not returned in results for the SearchHITs operation.
- Disposing of HITs can improve the performance of operations such as ListReviewableHITs and ListHITs.

Request Syntax

```
{
  "HITId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The ID of the HIT. Type: String	Yes

Response Elements

A successful request for the `DeleteHIT` operation returns with no errors and an empty body.

Example

The following example shows how to use the `DeleteHIT` operation:

Sample Request

The following example deletes a HIT with the specified HIT ID.

```
POST / HTTP/1.1
```

```
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  HITId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

DeleteQualificationType

Description

The `DeleteQualificationType` disposes a Qualification type and disposes any HIT types that are associated with the Qualification type.

This operation does not revoke Qualifications already assigned to Workers because the Qualifications might be needed for active HITs. If there are any pending requests for the Qualification type, Amazon Mechanical Turk rejects those requests. After you delete a Qualification type, you can no longer use it to create HITs or HIT types.

> **Note**
> DeleteQualificationType must wait for all the HITs that use the deleted Qualification type to be deleted before completing. It may take up to 48 hours before DeleteQualificationType completes and the unique name of the Qualification type is available for reuse with CreateQualificationType.

Request Syntax

```
{
  "QualificationTypeId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the QualificationType to dispose. Type: String	Yes

Response Elements

A successful request for the `DeleteQualificationType` operation returns with no errors and an empty body.

Example

The following example shows how to use the `DeleteQualificationType` operation:

Sample Request

The following example deletes a Qualification type and any HIT types that are associated with the Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
```

```
X-Amz-Date: <Date>
{
  QualificationTypeId:"AZ34EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

DeleteWorkerBlock

Description

The `DeleteWorkerBlock` operation allows you to reinstate a blocked Worker to work on your HITs. This operation reverses the effects of the CreateWorkerBlock operation. You need the Worker ID to use this operation. If the Worker ID is missing or invalid, this operation fails and returns the message “WorkerId is invalid.” If the specified Worker is not blocked, this operation returns successfully.

Request Syntax

```
{
  "WorkerId": String,

  "Reason": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`WorkerId`	The ID of the Worker to unblock Type: String	Yes
`Reason`	A message that explains the reason for unblocking the Worker. The Worker does not see this message. Type: String	No

Response Elements

A successful request for the `DeleteWorkerBlock` operation returns with no errors and an empty body.

DisassociateQualificationFromWorker

Description

The `DisassociateQualificationFromWorker` revokes a previously granted Qualification from a user.

You can provide a text message explaining why the Qualification was revoked. The user who had the Qualification can see this message.

Request Syntax

```
{
  "QualificationTypeId": String,

  "WorkerId": String,

  "IntegerValue": Integer,

  "Reason": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the Qualification type of the Qualification to be revoked. Type: String	Yes
`WorkerId`	The ID of the Worker who possesses the Qualification to be revoked. Type: String	Yes
`IntegerValue`	The value of the Qualification to assign. Type: Integer Default: 1	Yes
`Reason`	A text message that explains why the Qualification was revoked. The user who had the Qualification sees this message. Type: String	No

Response Elements

A successful request for the `DisassociateQualificationFromWorker` operation returns with no errors and an empty body.

Example

The following example shows how to use the `DisassociateQualificationFromWorker` operation:

Sample Request

The following example revokes Qualification of the specified Qualification type for the specified user.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  WorkerId:"AZ3456EXAMPLE",
  QualificationTypeId:"789RVWYBAZW00EXAMPLE"
  IntegerValue:1,

}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

GetAccountBalance

Description

The `GetAccountBalance` operation retrieves the amount of money in your Amazon Mechanical Turk account.

Request Syntax

```
{  }
```

Request Parameters

The request accepts the following data in JSON format:

Response Elements

A successful request returns a string representing your account balance details in US Dollars.

Example

The following example shows how to use the `GetAccountBalance` operation:

Sample Request

The following makes a GetAccountBalance request.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  AvailableBalance:10000.00
}
```

GetAssignment

Description

The `GetAssignment` retrieves an assignment with an AssignmentStatus value of Submitted, Approved, or Rejected, using the assignment's ID. Requesters can only retrieve their own assignments for HITs that they have not disposed of.

Request Syntax

```
{
  "AssignmentId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`AssignmentId`	The ID of the assignment you want to retrieve Type: String	Yes

Response Elements

A successful request returns an Assignment (p. 124) data structure.

GetFileUploadURL

Description

Important
Beginning Tuesday, December 12th 2017 the Answer Specification structure will **no longer support the `FileUploadAnswer` element** to be used for the QuestionForm (p. 97) data structure. Instead, we recommend that Requesters who want to create HITs asking Workers to upload files use Amazon S3.

The `GetFileUploadURL` operation generates and returns a temporary URL. You use the temporary URL to retrieve a file uploaded by a Worker as an answer to a FileUploadAnswer question for a HIT. The temporary URL is generated the instant the GetFileUploadURL operation is called, and is valid for 60 seconds. You can get a temporary file upload URL any time until the HIT is disposed. After the HIT is disposed, any uploaded files are deleted, and cannot be retrieved.

Request Syntax

```
{
  "AssignmentId": String,

  "QuestionIdentifier": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`AssignmentId`	The ID of the assignment that contains the question with a FileUploadAnswer. Type: String	Yes
`QuestionIdentifier`	The identifier of the question with a FileUploadAnswer, as specified in the QuestionForm of the HIT. Type: String	Yes

Response Elements

A successful request returns an FileUploadURL string containing the temporary URL.

GetHIT

Description

The `GetHIT` operation retrieves the details of the specified HIT.

Request Syntax

```
{
  "HITId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The ID of the HIT. Type: String	Yes

Response Elements

A successful request returns a HIT (p. 128) data structure.

GetQualificationScore

Description

The `GetQualificationScore` operation returns the value of a Worker's Qualification for a given Qualification type.

To get a Worker's Qualification, you must know the Worker's ID.

Only the owner of a Qualification type can query the value of a Worker's Qualification of that type.

Request Syntax

```
{
  "QualificationTypeId": String,

  "WorkerId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the QualificationType. Type: String	Yes
`WorkerId`	The ID of the Worker whose Qualification is being updated. Type: String	Yes

Response Elements

A successful request returns a Qualification (p. 40) data structure.

Example

The following example shows how to use the `GetQualificationScore` operation:

Sample Request

The following example disposes a Qualification type and any HIT types that are associated with the Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
```

```
  QualificationTypeId:"AZ34EXAMPLE",
  WorkerId:"AZ3456EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  WorkerId:"AZ3456EXAMPLE",
  IntegerValue:"95",
  GrantTime:"<date>"
}
```

GetQualificationType

Description

The `GetQualificationType`operation retrieves information about a Qualification type using its ID.

Request Syntax

```
{
  "QualificationTypeId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the QualificationType. Type: String	Yes

Response Elements

A successful request returns a QualificationType (p. 146) data structure.

Example

The following example shows how to use the `GetQualificationType` operation:

Sample Request

The following example gets information about a Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  QualificationTypeId:"AZ34EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
```

```
Date: <Date>
{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  Name:"EnglishWritingAbility",
  Description:"The ability to write and edit in text in English",
  QualificationTypeStatus:"Active"
}
```

ListAssignmentsForHIT

Description

The `ListAssignmentsForHIT` operation retrieves completed assignments for a HIT. You can use this operation to retrieve the results for a HIT.

You can get assignments for a HIT at any time, even if the HIT is not yet Reviewable. If a HIT requested multiple assignments, and has received some results but has not yet become Reviewable, you can still retrieve the partial results with this operation.

Use the AssignmentStatuses parameter to control which set of assignments for a HIT are returned. The GetAssignmentsForHIT operation can return submitted assignments awaiting approval, or it can return assignments that have already been approved or rejected. You can set AssignmentStatuses=Approved,Rejected to get assignments that have already been approved and rejected together in one result set.

Only the Requester who created the HIT can retrieve the assignments for that HIT.

Results are sorted and divided into numbered pages and the operation returns a single page of results. You can use the parameters of the operation to control sorting and pagination.

Request Syntax

```
{
  "HITId": String,

  "AssignmentStatuses": String,

  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The ID of the HIT. Type: String	Yes
`AssignmentStatuses`	The status of the assignments to return: Submitted | Approved | Rejected Type: String	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of Assignment (p. 124) data structures submitted for the HIT.

Example

The following example shows how to use the `ListAssignmentsForHIT` operation:

Sample Request

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  Assignments:[Assignment (p. 124)]
}
```

ListBonusPayments

Description

The `ListBonusPayments` operation retrieves the amounts of bonuses you have paid to Workers for a given HIT or assignment.

Request Syntax

```
{
  "HITId": String,

  "AssignmentId": String,

  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The ID of the HIT associated with the bonus payments to retrieve. If not specified, all bonus payments for all assignments for the given HIT are returned. Either the HITId parameter or the AssignmentId parameter must be specified Type: String	Conditional
`AssignmentId`	The ID of the assignment associated with the bonus payments to retrieve. If specified, only bonus payments for the given assignment are returned. Either the HITId parameter or the AssignmentId parameter must be specified Type: String	Conditional
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of Bonuses with the following fields: WorkerId, BonusAmount, AssignmentId, Reason and GrantTime

Example

The following example shows how to use the `ListBonusPayments` operation:

Sample Request

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  BonusPayments:[Bonus]
}
```

ListHITs

Description

The `ListHITs` operation returns all of a Requester's HITs. The operation returns HITs of any status, except for HITs that have been deleted of with the `DeleteHIT` operation or that have been auto-deleted.

Having high volumes of active HITs may lead to latency or timeouts when calling `ListHITs`. To remedy this, call the `DeleteHIT` (p. 29) operation on HITs you no longer need access to.

Request Syntax

```
{
  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of HIT (p. 128) data structures.

Example

The following example shows how to use the `ListHITs` operation:

Sample Request

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  HITs:[HIT (p. 128)]
}
```

ListHITsForQualificationType

Description

The `ListHITsForQualificationType` operation returns the HITs that use the given `QualififcationType` (p. 146) for a `QualificationRequirement` (p. 138). The operation returns HITs of any status, except for HITs that have been deleted with the DeleteHIT operation or that have been auto-deleted.

Having high volumes of active HITs may lead to latency or timeouts when calling `ListHITsForQualificationType`. To remedy this, call the `DeleteHIT` (p. 29) operation on HITs you no longer need access to.

Request Syntax

```
{
  "QualificationTypeId": String,

  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the Qualification type to use when querying HITs. Type: String	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of HIT (p. 128) data structures.

Example

The following example shows how to use the `ListHITsForQualificationType` operation:

Sample Request

```
POST / HTTP/1.1
```

```
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  HITs:[HIT (p. 128)]
}
```

ListQualificationRequests

Description

The `ListQualificationRequests` operation retrieves requests for Qualifications of a particular Qualification type. The owner of the Qualification type calls this operation to poll for pending requests, and accepts them using the AcceptQualification operation.

Request Syntax

```
{
  "QualificationTypeId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the QualificationType. Type: String	No

Response Elements

A successful request returns a paginated list of QualificationRequests.

Example

The following example shows how to use the `ListQualificationRequests` operation:

Sample Request

The following example lists requests for a Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  QualificationTypeId:"AZ34EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
```

```
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  QualificationRequests:[QualificationRequest (p. 136) ],
  NumResults:10,
  NextToken:PaginationToken
}
```

ListQualificationTypes

Description

The `ListQualificationTypes` operation searches for Qualification types using the specified search query, and returns a list of Qualification types.

Request Syntax

```
{
  "Query": String,

  "MustBeRequestable": Boolean,

  "MustBeOwnedByCaller": Boolean,

  "NextToken": String,

  "MaxResults": Integer,
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Query`	A search term Type: String	No
`MustBeRequestable`	Specifies that only Qualification types that a user can request through the Amazon Mechanical Turk web site, such as by taking a Qualification test, are returned as results of the search. Some Qualification types, such as those assigned automatically by the system, cannot be requested directly by users. If false, all Qualification types, including those managed by the system, are considered for the search. Type: Boolean	Yes
`MustBeOwnedByCaller`	Specifies that only Qualification types that the Requester created are returned. If false, the operation returns all Qualification types. Type: Boolean Default: False	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of Qualification Types.

Example

The following example shows how to use the `ListQualificationTypes` operation:

Sample Request

The following example performs a simple text query for Qualification types.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  Query:"LanguageSkill"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  QualificationType:[QualificationType (p. 146)]
}
```

ListReviewableHITs

Description

The `ListReviewableHITs` operation returns all of a Requester's HITs that have not been approved or rejected. The operation can return any HITs that have not been deleted with the DeleteHIT operation or that have not been auto-deleted.

By default, `ListReviewableHITs` operation only returns the IDs of HITs in the status Reviewable.

If you used the UpdateHITReviewStatus (p. 73) Operation to set the status of the HIT to Reviewing, and want HITs in the status Reviewing, set the Status request parameter to `Reviewing`.

Request Syntax

```
{
  "HITTypeId": String,

  "Status": String,

  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITTypeId`	The ID of the HIT type of the HITs to consider for the query. Type: String	No
`Status`	The status of the HITs to return: Reviewable | Reviewing Type: String By Default Status is set to Reviewable.	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of HIT (p. 128) data structures.

Example

The following example shows how to use the `ListReviewableHITs` operation:

Sample Request

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  HITs:[HIT (p. 128)]
}
```

ListReviewPolicyResultsForHIT

Description

The `ListReviewPolicyResultsForHIT` operation retrieves the computed results and the actions taken in the course of executing your Review Policies during a CreateHIT operation. For information about how to apply Review Policies when you call CreateHIT, see Review Policies. The GetReviewResultsForHIT operation can return results for both Assignment-level and HIT-level review results. You can also specify to only return results pertaining to a particular Assignment.

Request Syntax

```
{
  "HITId": String,

  "PolicyLevel": String,

  "AssignmentId": String,

  "RetrieveActions": String,

  "RetrieveResults": String,

  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The unique identifier of the HIT to retrieve review results for. Type: String	Yes
`PolicyLevel`	The Policy Level(s) to retrieve review results for - HIT or Assignment. If omitted, the default behavior is to retrieve all data for both policy levels. For a list of all the described policies, see Review Policies. Type: String	Yes
`AssignmentId`	If supplied, the results are limited to those pertaining directly to this Assignment ID. Type: String	No
`RetrieveActions`	Retrieves a list of the actions taken executing the Review Policies and their outcomes. T or F. Type: String	No

Name	Description	Required
`RetrieveResults`	Retrieves a list of the results computed by the Review Policies. T or F. Type: String	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request operation has a ListReviewPolicyResultsForHITResponse element in the response. The ListReviewPolicyResultsForHITResponse element contains the name of the Review Policy applied as well as the AssignmentReviewReport element and the HITReviewReport element.

ListWorkerBlocks

Description

The `ListWorkersBlocks` operation retrieves a list of Workers who are blocked from working on your HITs.

Request Syntax

```
{
  "NextToken": String,

  "MaxResults": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of Workers that have been blocked along with the reason for the block.

Example

The following example shows how to use the `ListWorkerBlocks` operation:

Sample Request

The following example lists all Worker blocks.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  WorkerBlocks:[{WorkerId, Reason}],
  NumResults:3
}
```

ListWorkersWithQualificationType

Description

The `ListWorkersWithQualificationType` operation returns all of the Workers with a given Qualification type.

Request Syntax

```
{
  "QualificationTypeId": String,

  "Status": String,

  "NextToken": String,

  "MaxResults": Integer,
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationTypeId`	The ID of the QualificationType. Type: String	No
`Status`	The status of the Qualifications to return. Granted | Revoked Type: String	No
`NextToken`	Pagination token Type: String	No
`MaxResults`	Type: Integer	No

Response Elements

A successful request returns a paginated list of Qualifications that have been granted to Workers.

Example

The following example shows how to use the `ListWorkersWithQualificationType` operation:

Sample Request

The following example performs a simple text query for Qualification types.

```
POST / HTTP/1.1
```

```
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  QualificationTypeId:"ZSPJXD4F1SFZP7YNJWR0"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  NextToken:PaginationToken,
  NumResults:10,
  Qualifications:[Qualification (p. 134)]
}
```

NotifyWorkers

Description

The `NotifyWorkers` operation sends an email to one or more Workers that you specify with the Worker ID. You can specify up to 100 Worker IDs to send the same message with a single call to the NotifyWorkers operation. The NotifyWorkers operation will send a notification email to a Worker only if you have previously approved or rejected work from the Worker.

Request Syntax

```
{
  "Subject": String,

  "MessageText": String,

  "WorkerIds": Array of Strings
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Subject`	The subject line of the email message to send. Can include up to 200 characters. Type: String	Yes
`MessageText`	The text of the email message to send. Can include up to 4,096 characters. Type: String	Yes
`WorkerIds`	An array of WorkerIds to notify. You can notify upto 100 Workers at a time. Type: Array of Strings	Yes

Response Elements

A successful request for the `NotifyWorkers` operation returns with no errors and an empty body.

RejectAssignment

Description

The `RejectAssignment` operation rejects the results of a completed assignment.

You must include a feedback message with the rejection, which the Worker can see in the Status section of the web site. When you include a feedback message with the rejection, it helps the Worker understand why the assignment was rejected, and can improve the quality of the results the Worker submits in the future.

Only the Requester who created the HIT can reject an assignment for the HIT.

Note
To maintain the consistency of the assignments for HITs in a batch, if a HIT is created through the Amazon Mechanical Turk Requester website, you must use the Requester website UI to approve or reject the work.
If you want to be able to approve or reject work through the API, you can use a HITLayout (p. 86) when calling CreateHIT (p. 12) to utilize an existing template from the Amazon Mechanical Turk Requester website

Request Syntax

```
{
  "AssignmentId": String,

  "RequesterFeedback": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`AssignmentId`	The ID of the assignment. This parameter must correspond to a HIT created by the Requester. Type: String	Yes
`RequesterFeedback`	A message for the Worker, which the Worker can see in the Status section of the web site. Type: String Constraints: Can be up to 1024 characters (including multi-byte characters). The RequesterFeedback parameter cannot contain ASCII characters 0-8, 11,12, or 14-31. If these characters are present, the operation throws an InvalidParameterValue error.	Yes

Response Elements

A successful request for the `RejectAssignment` operation returns with no errors and an empty body.

Example

The following example shows how to use the `RejectAssignment` operation:

Sample Request

The following example rejects an assignment identified by its assignment ID.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  AssignmentId:"123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

RejectQualificationRequest

Description

The `RejectQualificationRequest` operation rejects a user's request for a Qualification.

You can provide a text message explaining why the request was rejected. The Worker who made the request can see this message.

Request Syntax

```
{
  "QualificationRequestId": String,

  "Reason": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`QualificationRequestId`	The ID of the Qualification request, as returned by the ListQualificationRequests (p. 52) operation. Type: String	Yes
`Reason`	A text message explaining why the request was rejected, to be shown to the Worker who made the request. Type: String	No

Response Elements

A successful request for the `RejectQualificationRequest` operation returns with no errors and an empty body.

Example

The following example shows how to use the `RejectQualificationRequest` operation:

Sample Request

The following example rejects a specified Qualification request.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
```

```
{
  QualificationRequestId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

SendBonus

Description

The `SendBonus` operation issues a payment of money from your account to a Worker. This payment happens separately from the reward you pay to the Worker when you approve the Worker's assignment. The SendBonus operation requires the Worker's ID and the assignment ID as parameters to initiate payment of the bonus. You must include a message that explains the reason for the bonus payment, as the Worker may not be expecting the payment. Amazon Mechanical Turk collects a fee for bonus payments, similar to the HIT listing fee.

This operation fails if your account does not have enough funds to pay for both the bonus and the fees. This operation may also fail if the Worker in question has not completed an Assignment for you in the last six months.

Request Syntax

```
{
  "WorkerId": String,

  "AssignmentId": String,

  "BonusAmount": String,

  "Reason": String,

  "UniqueRequestToken": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`WorkerId`	The ID of the Worker being paid the bonus. Type: String	Yes
`AssignmentId`	The ID of the assignment for which this bonus is paid. Type: String	Yes
`BonusAmount`	The bonus is specified as a US Dollar amount. Type: String	Yes
`Reason`	A message that explains the reason for the bonus payment. The Worker receiving the bonus can see this message. Type: String	Yes
`UniqueRequestToken`	A unique identifier for this request, which allows you to retry the call on error without granting multiple bonuses. This is useful in cases such as network	No

Name	Description	Required
	timeouts where it is unclear whether or not the call succeeded on the server. If the bonus already exists in the system from a previous call using the same UniqueRequestToken, subsequent calls will return an error with a message containing the request ID. Type: String Constraints: must not be longer than 64 characters in length.	

Response Elements

A successful request for the `SendBonus` operation returns with no errors and an empty body.

SendTestEventNotification

Description

The `SendTestEventNotification` operation causes Amazon Mechanical Turk to send a notification message as if a HIT event occurred, according to the provided notification specification. This allows you to test notifications without setting up notifications for a real HIT type and trying to trigger them using the website. When you call this operation, the service sends the test notification immediately.

Request Syntax

```
{
  "Notification": Notification data structure,

  "TestEventType": An EventType element of the Notification data structure.
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`Notification (p. 174)`	The notification specification to test. This value is identical to the value you would provide to the UpdateNotificationSettings operation when you establish the notification specification for a HIT type. Type: Notification data structure	Yes
`TestEventType`	The event to simulate to test the notification specification. This event is included in the test message even if the notification specification does not include the event type. The notification specification does not filter out the test event. Type: An EventType element of the Notification data structure.	Yes

Response Elements

A successful request for the `SendTestEventNotification` operation returns with no errors and an empty body.

UpdateExpirationForHIT

Description

The `UpdateExpirationForHIT` operation allows you extend the expiration time of a HIT beyond is current expiration or expire a HIT immediately. You cannot shorten the expiration time so that you're not affecting Workers who have accepted your HIT.

To expire a HIT immediately, provide the value 0 or set ExpireAt to a time in the past.

Request Syntax

```
{
  "HITId": String,

  "ExpireAt": Timestamp
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The HIT to update. Type: String	Yes
`ExpireAt`	The date and time at which you want the HIT to expire Type: Timestamp	Yes

Response Elements

A successful request for the `UpdateExpirationForHIT` operation returns with no errors and an empty body.

UpdateHITReviewStatus

Description

The `UpdateHITReviewStatus` operation toggles the status of a HIT. If the status is Reviewable, this operation updates the status to Reviewing, or reverts a Reviewing HIT back to the Reviewable status.

For example, when processing assignments from Workers if you do not want to make an immediate decision about approving or rejecting assignments, you can use this operation to set the status of the HIT to Reviewing . To retrieve a list of HITs in reviewing status, add `"Status": "Reviewing"` to your request parameters in the ListReviewableHITs (p. 56) Operation.

Request Syntax

```
{
  "HITId": String,

  "Revert": Boolean
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The HIT to update. Type: String	Yes
`Revert`	Specifies whether to update the HIT Status from Reviewing to Reviewable. Type: Boolean Default: false; the operation promotes the HIT from Reviewable to Reviewing.	No

Response Elements

A successful request for the `UpdateHITReviewStatus` operation returns with no errors and an empty body.

UpdateHITTypeOfHIT

Description

The `UpdateHITTypeOfHIT` operation allows you to change the HITType properties of a HIT. This operation disassociates the HIT from its old HITType properties and associates it with the new HITType properties. The HIT takes on the properties of the new HITType in place of the old ones.

Request Syntax

```
{
  "HITId": String,

  "HITTypeId": String
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITId`	The HIT to update. Type: String	Yes
`HITTypeId`	The ID of the new HIT type. Type: String	Yes

Response Elements

A successful request for the `UpdateHITTypeOfHIT` operation returns with no errors and an empty body.

UpdateNotificationSettings

Description

The `UpdateNotificationSettings` operation creates, updates, disables or re-enables notifications for a HIT type.

If you call the UpdateNotificationSettings operation for a HIT type that already has a notification specification, the operation replaces the old specification with a new one.

You can call the UpdateNotificationSettings operation to enable or disable notifications for the HIT type, without having to modify the notification specification itself.

You can call this operation at any time to change the value of the of the Active parameter of a HIT type. You can specify changes to the Active status without specifying a new notification specification (the Notification parameter).

To change the Active status of a HIT type's notifications, the HIT type must already have a notification specification, or one must be provided in the same call to UpdateNotificationSettings.

Request Syntax

```
{
  "HITTypeId": String,

  "Notification": Notification (p. 174) data structure,

  "Active": Boolean
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
`HITTypeId`	The the HITTypeID whose notification specification is being updated. Type: String	Yes
`Notification`	The notification specification for the HIT type. Type: Notification (p. 174) data structure	Conditional
`Active`	Specifies whether notifications are sent for HITs of this HIT type, according to the notification specification. You must specify either the Notification parameter or the Active parameter for the call to SetHITTypeNotification to succeed. Type: Boolean	Conditional

Response Elements

A successful request for the `UpdateNotificationSettings` operation returns with no errors and an empty body.

UpdateQualificationType

Description

The `UpdateQualificationType` operation modifies the attributes of an existing Qualification type, which is represented by a QualificationType data structure. Only the owner of a Qualification type can modify its attributes.

Most attributes of a Qualification type can be changed after the type has been created. However, the Name and Keywords fields cannot be modified. The RetryDelayInSeconds parameter can be modified or added to change the delay or to enable retries, but RetryDelayInSeconds cannot be used to disable retries.

You can use this operation to update the test for a Qualification type. The test is updated based on the values specified for the Test, TestDurationInSeconds and AnswerKey parameters. All three parameters specify the updated test. If you are updating the test for a type, you must specify the Test and TestDurationInSeconds parameters. The AnswerKey parameter is optional; omitting it specifies that the updated test does not have an answer key.

If you omit the Test parameter, the test for the Qualification type is unchanged. There is no way to remove a test from a Qualification type that has one. If the type already has a test, you cannot update it to be AutoGranted. If the Qualification type does not have a test and one is provided by an update, the type will henceforth have a test.

If you want to update the test duration or answer key for an existing test without changing the questions, you must specify a Test parameter with the original questions, along with the updated values.

If you provide an updated Test but no AnswerKey, the new test will not have an answer key. Requests for such Qualifications must be granted manually.

You can also update the AutoGranted and AutoGrantedValue attributes of the Qualification type.

Request Syntax

```
{
  "QualificationTypeId": String,

  "RetryDelayInSeconds": Integer,

  "QualificationTypeStatus": String,

  "Description": String,

  "Test": String,

  "AnswerKey": String,

  "TestDurationInSeconds": Integer,

  "AutoGranted": Boolean,

  "AutoGrantedValue": Integer
 }
```

Request Parameters

The request accepts the following data in JSON format:

Name	Description	Required
QualificationTypeId	The ID of the Qualification type to update. Type: String	Yes
RetryDelayInSeconds	The amount of time, in seconds, that Workers must wait after requesting a Qualification of the specified Qualification type before they can retry the Qualification request. It is not possible to disable retries for a Qualification type after it has been created with retries enabled. If you want to disable retries, you must dispose of the existing retry-enabled Qualification type using DisposeQualificationType and then create a new Qualification type with retries disabled using CreateQualificationType. Type: Integer	No
QualificationTypeStatus	The new status of the Qualification type - Active | Inactive Type: String	No
Description	The new description of the Qualification type. Type: String	No
Test	The questions for the Qualification test a Worker must answer correctly to obtain a Qualification of this type. If this parameter is specified, TestDurationInSeconds must also be specified. Type: String Constraints: Must not be longer than 65535 bytes. Must be a QuestionForm data structure. This parameter cannot be specified if AutoGranted is true. Default: None. If not specified, the Worker may request the Qualification without answering any questions.	No
AnswerKey	The answers to the Qualification test specified in the Test parameter, in the form of an AnswerKey data structure. Type: String Constraints: Must not be longer than 65535 bytes. Default: None. If not specified, you must process Qualification requests manually.	No
TestDurationInSeconds	The number of seconds the Worker has to complete the Qualification test, starting from the time the Worker requests the Qualification. This is required if the Test parameter is specified.	Conditional

Name	Description	Required
	Type: Integer	
AutoGranted	Specifies whether requests for the Qualification type are granted immediately, without prompting the Worker with a Qualification test. Type: Boolean Constraints: If the Test parameter is specified, this parameter cannot be true. Default: False	No
AutoGrantedValue	The Qualification value to use for automatically granted Qualifications. This parameter is used only if the AutoGranted parameter is true. Type: Integer Default: If AutoGranted is true, AutoGrantedValue is set to 1.	No

Response Elements

A successful request returns a QualificationType (p. 146) data structure.

Example

The following example shows how to use the `UpdateQualificationType` operation:

Sample Request

The following example changes the QualificationTypeStatus of a Qualification type.

```
POST / HTTP/1.1
Host: mturk-requester.us-east-1.amazonaws.com
Content-Length: <PayloadSizeBytes>
X-Amz-Date: <Date>
{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  QualificationTypeStatus:"Inactive"
}
```

Sample Response

The following is an example response:

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
Content-Type: application/x-amz-json-1.1
Content-Length: <PayloadSizeBytes>
Date: <Date>
{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  Name:"EnglishWritingAbility",
```

```
  Description:"The ability to write and edit in text in English",
  QualificationTypeStatus:"Inactive"
}
```

Question and Answer Data

Topics

The questions and answers that Amazon Mechanical Turk passes between Requesters and Workers are XML documents that conform to schemas. These documents are passed to the service and returned by the service as parameter values.

Crowd HTML Elements

Topics

Description

Crowd HTML Elements extend the capabilities of the ExternalQuestion (p. 92) and HTMLQuestion (p. 88). They are web components that provide a number of task widgets and design elements that can be tailored to the question being asked.

Use Cases

Crowd HTML Elements are web components, a web standard that abstracts HTML markup, CSS, and JavaScript functionality into an HTML tag or set of tags. If you'd like to see how that works, try the `<crowd-bounding-box>` example below. The element provides a bounding box widget which can be customized with different instructions, labels, and headers.

Other types of questions that can be customized include: semantic segmentation, image classification, text classification, utterance collection, and more.

Examples

To quickly try one of the examples below, open a text editor on your local machine, copy an example from this page, paste it into the text editor, and then save the file with whatever name you want and a `.html` extension. Open the file in a browser and the example should work. You can try customizing it further or adding other **Crowd HTML Elements** by exploring the element reference.

Crowd HTML Essential Elements

Every use of **Crowd HTML Elements** requires two things:

- The **Crowd HTML Elements** loader, which is a `<script>` element that should be placed before your form.

  ```
  <script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
  ```

- Opening and closing `<crowd-form>` tags. Put your form's content between the tags. The benefit to these is that they set up the specifications for your form and the "submit" button. You write less code and need to remember fewer specific things.

Crowd HTML Bounding Box

Try this bounding box example. Copy it, paste it, and run it in a browser as instructed above.

Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

```
<crowd-form>
  <crowd-bounding-box
    name="annotatedResult"
    labels="['Basketball player', 'Referee']"
    src="https://s3.amazonaws.com/cv-demo-images/basketball-outdoor.jpg"
    header="Draw boxes around each basketball player and referee in this image"
  >
    <full-instructions header="Bounding Box Instructions" >
      <p>Use the bounding box tool to draw boxes around the requested target of interest:</
p>
      <ol>
        <li>Draw a rectangle using your mouse over each instance of the target.</li>
        <li>Make sure the box does not cut into the target, leave a 2 - 3 pixel margin</li>
        <li>
          When targets are overlapping, draw a box around each object,
          include all contiguous parts of the target in the box.
          Do not include parts that are completely overlapped by another object.
        </li>
        <li>
          Do not include parts of the target that cannot be seen,
          even though you think you can interpolate the whole shape of the target.
        </li>
        <li>Avoid shadows, they're not considered as a part of the target.</li>
        <li>If the target goes off the screen, label up to the edge of the image.</li>
      </ol>
    </full-instructions>

    <short-instructions>
      Draw boxes around each basketball player and referee in this image.
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

Pay attention to the attributes and regions of the `<crowd-bounding-box>` element. It requires the `header`, `labels`, `name`, and `src` attributes. It also requires the `<full-instructions>` and `<short-instructions>` regions, though what is put in them is up to you.

The sample will also display the form output when you press the "submit" button, so you can see the format of the output you will receive.

Crowd HTML Sentiment Analysis

Try this sentiment analysis example. Copy it, paste it, and run it in a browser as instructed above.

Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-classifier
      name="sentiment"
      categories="['Positive', 'Negative', 'Neutral', 'N/A']"
      header="What sentiment does this text convey?"
    >
      <classification-target>
        Everything is wonderful.
      </classification-target>

      <full-instructions header="Sentiment Analysis Instructions">
        <p><strong>Positive</strong> sentiments include: joy, excitement, delight</p>
        <p><strong>Negative</strong> sentiments include: anger, sarcasm, anxiety</p>
        <p><strong>Neutral</strong>: neither positive or negative, such as stating a fact</
p>
```

```
        <p><strong>N/A</strong>: when the text cannot be understood</p>
        <p>When the sentiment is mixed, such as both joy and sadness, use your judgment to
 choose the stronger emotion.</p>
      </full-instructions>

      <short-instructions>
       Choose the primary sentiment that is expressed by the text.
      </short-instructions>
    </crowd-classifier>
</crowd-form>
```

This shares some characteristics with other elements, like the bounding box above. For example, they both require a name, a header, and instructions. One thing that's notably different is the `<classification-target>` region. That has simple text in it, but it can contain just about any HTML: a video clip, an audio clip, an animation, anything that can be represented in a browser and simply classified.

Element Reference

The Custom HTML Element Reference provides a list of all supported custom elements, their requirements, attributes, and sample outputs (where appropriate).

Related Documents

- HTMLQuestion (p. 88)
- ExternalQuestion (p. 92)
- Element Reference

Using XML Parameter Values

The HTMLQuestion (p.),ExternalQuestion (p.), QuestionForm (p.), QuestionFormAnswers (p.), and AnswerKey (p.) data structures are used as parameter values in service requests, and as return values in service responses. Unlike other data structures described in this API reference, these XML structures are not part of the service API directly, but rather are used as string values going in and out of the service. This article describes the encoding methods needed to use XML data as parameter and return values.

XML Data as a Parameter

Data must be *URL encoded* to appear as a single parameter value in the request. Characters that are part of URL syntax, such as question marks (**?**) and ampersands (**&**), must be replaced with the corresponding URL character codes.

> **Note**
> XML data should only be URL encoded, *not* XML escaped.

In service responses, this data will be XML escaped.

Namespaces for XML Parameter Values

XML data in parameter values must have a namespace specified for all elements. The easiest way to do this is to include an `xmlns` attribute in the root element equal to the appropriate namespace.

The namespace for a `HTMLQuestion`,`ExternalQuestion`, `QuestionForm`, `QuestionFormAnswers`, or `AnswerKey` element is identical to the URL of the corresponding schema document, including the version date. While XML namespaces need not be URLs according to the XML specification, this convention ensures that the consumer of the value knows which version of the schema is being used for the data.

For the locations of the schema documents, as well as instructions on how to include the version date in the URL, see Schema Locations (p.).

HITLayout

Topics

Description

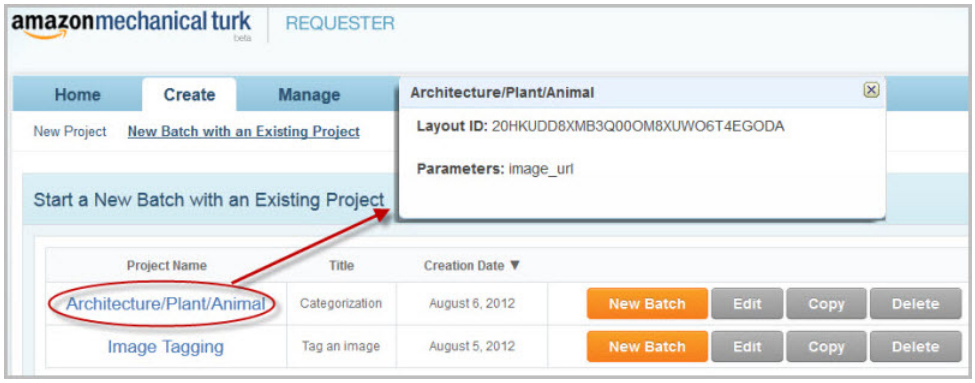
A HITLayout is a reusable Amazon Mechanical Turk project template used to provide Human Intelligence Task (HIT) question data for `CreateHIT`. You can create a HITLayout template by creating a Mechanical Turk project on the Amazon Mechanical Turk Requester website. For more information about creating a project, see How to Create a Project in the Requester UI Guide.

Obtaining a Layout ID

A **Layout ID** is assigned to each Mechanical Turk project you create on the Requester website. You use the **Layout ID** as the value for `HITLayoutId` when calling `CreateHIT` to identify the HITLayout project template to use. Mechanical Turk projects can contain parameter placeholders in the format `${parameter_name}`. The names for the parameter placeholders used in a HITLayout project template are listed as **Parameters** along with the **Layout ID** on the Requester website.

To view the Layout ID and the Parameters used in your HITLayout project template

1. Go to the Amazon Mechanical Turk Requester website. Or for the Requester Sandbox site, go to the Amazon Mechanical Turk Requester Sandbox website.
2. Click **Create**, and then click **New Batch with an Existing Project**.
3. Click the **Project Name** of an existing project to view **Layout ID** and **Parameters**.



Using a HITLayout

You can use the HITLayout form of a HIT by calling `CreateHIT` with a `HITLayoutId` and a list of HITLayoutParameter (p. 133) structures. The project parameter placeholders are replaced with values from the HITLayoutParameter (p. 133) structures when you call `CreateHIT` to create a HIT. You need one structure for each of the parameter values you want substituted. The parameter names that you pass to `CreateHIT` must match the parameter names used in the HITLayout project template created on the Requester website. The parameter values cannot be changed after the HIT has been created.

Note
You can use either the `HITLayoutId` or the `Question` parameter when calling `CreateHIT`, but not both.

Each `CreateHIT` call merges the parameter values from HITLayoutParameter structures into the HITLayout template to generate the HIT question document. You use the same **Layout ID** in `HITLayoutId` to call `CreateHIT` multiple times, with different parameter values supplied each time for the placeholders.

Requesters can use this parameter substitution capability to create a large number of HITs that all share a common design. For example, you can create a HIT question that asks Workers to provide keywords for an image and draw boxes around key image features using a JavaScript library. First, you use the Requester website to create a Mechanical Turk project that uses a parameter placeholder for the image URL. Then you call `CreateHIT` using the same HITLayout template iteratively, using a different image URL value each time. Each call to `CreateHIT` uses the same **Layout ID**, but each call uses a different HITLayoutParameter structure that contains a unique image URL.

Guidelines for Using HITLayouts

- After a HIT is created, the HIT behaves like an HTMLQuestion HIT, which gives you the option to use HTML and JavaScript features in your HIT design, including Asynchronous JavaScript and XML (AJAX) callbacks.
- Parameter substitution allows you to replace a short parameter name with long strings of text. You will receive errors if the resulting document is longer than permitted by the `Question` parameter of `CreateHIT`.
- The HITLayout is used to create an HTMLQuestion document. HITLayoutParameter values with reserved characters or invalid HTML markup may result in an invalid HTMLQuestion document. For more information, see HTMLQuestion (p. 88).

HTMLQuestion

Topics

Description

The `HTMLQuestion` data structure defines one or more questions for a HIT using HTML. The `HTMLQuestion` data structure is similar to both the `QuestionForm` and `ExternalQuestion` data structures.

The `QuestionForm` data structure defines, using a special XML language, how Amazon Mechanical Turk displays HIT questions and collects the answers. The `ExternalQuestion` data structure defines, using HTML, questions you host on your own "external" website. If you want to define your questions using HTML forms without having to host a website, you can use the `HTMLQuestion` data structure.

A `HTMLQuestion` HIT is like a cross between a `QuestionForm` HIT and an `ExternalQuestion` HIT, for instance:

- Like a `QuestionForm` HIT, you do not need to run a website or run any other infrastructure to have your HIT display on Mechanical Turk. You define your question when you call CreateHIT and then collect worker answers later, after they have been submitted.
- Like an `ExternalQuestion` HIT, you can define your HIT in HTML. Your HTML code must contain a form for the Worker to fill out and submit, which is displayed in a frame in the Worker's web browser. The Worker submits results using your form, and your form submits the results back to Amazon Mechanical Turk. Worker answers are processed by Mechanical Turk in the same way as `ExternalQuestion` HITs. If you choose, you can collect or process the results before submitting to Mechanical Turk.

The worker interaction and presentation options available for `HTMLQuestion` are similar to `ExternalQuestion`. HTMLQuestions differ from ExternalQuestions primarily in how they are created.

As with the other question data structures, an `HTMLQuestion` is a string value that consists of XML data. This data must conform to the `HTMLQuestion` schema. See Data Structure Schema Locations (p. 122) for the location of this schema. For more information about using XML data as a parameter or return value, see Using XML Parameter Values (p. 85).

> **Note**
> You can only use an `HTMLQuestion` as the question of a HIT. You cannot use an `HTMLQuestion` with a Qualification test.

The `HTMLQuestion` data structure is used as a parameter value for the following operation:

- `CreateHIT`

The `HTMLQuestion` data structure is a value in a HIT (p. 128) data structure.

All elements in an `HTMLQuestion` belong to a namespace whose name is identical to the URL of the `HTMLQuestion` schema document for the version of the API you are using.

The HTMLQuestion Data Structure

The `HTMLQuestion` data structure has a root element of `HTMLQuestion`.

The `HTMLQuestion` element contains the following elements:

Name	Description	Required
`HTMLContent`	The HTML code of your web form, to be displayed in a frame in the Worker's web browser. The HTML must validate against the HTML5 specification. HTML5 is backwards-compatible with a variety of recent HTML document specifications. For more information, see http://www.w3.org/TR/html5-diff/. For help in ensuring that your HTML validates, see http://validator.w3.org. Type: String Default: None Amazon Mechanical Turk appends the following parameters to this URL: `assignmentId`, `hitId`, `turkSubmitTo`, and `workerId`. For more information about these appended parameters, see the sections following this table.	Yes
`FrameHeight`	The height of the frame, in pixels. If you set the value to 0, your HIT will automatically resize to fit within the Worker's browser window. Type: Integer Default: None	Yes

Example

The following is an example of a complete `HTMLQuestion` data structure. Remember that to pass this structure in as the value of a parameter to an operation, XML characters must be escaped as character entities. For more information, see Using XML Parameter Values (p. 85).

```
<HTMLQuestion xmlns="[the HTMLQuestion schema URL]">
  <HTMLContent><![CDATA[
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>
  <script type='text/javascript' src='https://s3.amazonaws.com/mturk-public/
externalHIT_v1.js'></script>
 </head>
 <body>
  <form name='mturk_form' method='post' id='mturk_form' action='https://www.mturk.com/
mturk/externalSubmit'>
  <input type='hidden' value='' name='assignmentId' id='assignmentId'/>
```

```
  <h1>What's up?</h1>
  <p><textarea name='comment' cols='80' rows='3'></textarea></p>
  <p><input type='submit' id='submitButton' value='Submit' /></p></form>
  <script language='Javascript'>turkSetAssignmentID();</script>
 </body>
</html>
]]>
  </HTMLContent>
  <FrameHeight>0</FrameHeight>
</HTMLQuestion>
```

Using Crowd HTML Elements

The HTML Question supports Crowd HTML Elements (p. 82). Based on HTML web components, they encapsulate HTML, CSS, and JavaScript functionality behind a single HTML element. For example, the `<crowd-form>` element sets up your form for you, setting the correct submission endpoint and inserting a submit button at the end. Other elements provide question widgets or design elements you can customize to create more tailored HIT structures.

Try this sample sentiment analysis form.

Example

```
<HTMLQuestion xmlns="http://mechanicalturk.amazonaws.com/
AWSMechanicalTurkDataSchemas/2011-11-11/HTMLQuestion.xsd">
  <HTMLContent><![CDATA[
    <!DOCTYPE html>
      <body>
        <script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
        <crowd-form>
          <crowd-classifier
            name="sentiment"
            categories="['Positive', 'Negative', 'Neutral', 'N/A']"
            header="What sentiment does this text convey?"
          >
            <classification-target>
            Everything is wonderful.
            </classification-target>

            <full-instructions header="Sentiment Analysis Instructions">
            <p><strong>Positive</strong>
              sentiment include: joy, excitement, delight</p>
            <p><strong>Negative</strong> sentiment include:
              anger, sarcasm, anxiety</p>
            <p><strong>Neutral</strong>: neither positive or
              negative, such as stating a fact</p>
            <p><strong>N/A</strong>: when the text cannot be
              understood</p>
            <p>When the sentiment is mixed, such as both joy and sadness,
              use your judgment to choose the stronger emotion.</p>
            </full-instructions>

            <short-instructions>
             Choose the primary sentiment that is expressed by the text.
            </short-instructions>
          </crowd-classifier>
        </crowd-form>
      </body>
    </html>
  ]]></HTMLContent>
  <FrameHeight>0</FrameHeight>
</HTMLQuestion>
```

Between the <classification-target> opening and closing tags, you can put any HTML that could be rendered and classified. For example you could use an audio clip or a video clip.

For more details, read the Crowd HTML Elements (p. 82) article or view the Element Reference to see the different elements that are available.

Preview Mode

The question defined by `HTMLQuestion` displays when a Worker previews the HIT on the Amazon Mechanical Turk website, before the Worker clicks the **Accept HIT** button. When the HIT is being previewed, the URL has a special value for the `assignmentId`: `ASSIGNMENT_ID_NOT_AVAILABLE`. This is the same mechanism used for `ExternalQuestion` HITs.

When a Worker previews a HIT, your HTML should show the Worker everything they will need to do to complete the HIT, so they can decide whether or not to accept it. The easiest way to do this is to simply display the form as it would appear when the HIT is accepted. However, you may want to take precautions to prevent a Worker from accidentally filling out or submitting your form prior to accepting the HIT.

You can use JavaScript to check the `assignmentId` parameter, and change the display of the form if the HIT is being previewed (`assignmentId=ASSIGNMENT_ID_NOT_AVAILABLE`).

The Form Action

For information about form actions for `HTMLQuestion`, see " The Form Action" in ExternalQuestion (p. 92).

The Answer Data

For information about answer data for `HTMLQuestion`, see " The Answer Data" in ExternalQuestion (p. 92).

Guidelines For Using HTML Questions

Tip
Your HTML code can do many things inside the browser frame, but eventually it must cause the Worker's browser to load the "externalSubmit" URL in the frame with the results in POST data. The easiest way to do this is with an HTML form whose fields contain the HIT results, with a submit button that the Worker clicks. If a `HTMLQuestion` HIT prevents the Worker from submitting results back to Amazon Mechanical Turk using the "externalSubmit" mechanism, the Worker may not be able to claim rewards or continue doing work without restarting their session. Amazon Mechanical Turk reserves the right to remove any `HTMLQuestion` HITs that are not functioning properly.

Note
Your HIT will be rendered inside an IFRAME that has certain limitations. The IFRAME operates in HTML5 “sandbox” mode that has extra restrictions on the content that can appear in the frame. This limits your ability to execute certain code and to use technologies such as Adobe Flash. To ensure your HITs work as expected, we recommend you test them first in the Requester Sandbox.

Tip
All `HTMLQuestion` HITs are served from the same domain, regardless of requester. Bear this in mind if you choose to set cookies from JavaScript in your HTML.

ExternalQuestion

Topics

Description

Instead of providing a QuestionForm data structure (p. 97) that tells Amazon Mechanical Turk how to display your questions and collect answers, you can host the questions on your own website using an "external" question.

A HIT with an external question displays a web page from your website in a frame in the Worker's web browser. Your web page displays a form for the Worker to fill out and submit. The Worker submits results using your form, and your form submits the results back to Amazon Mechanical Turk. Using your website to display the form gives your website control over how the question appears and how answers are collected.

To use an external question with a HIT, you provide an `ExternalQuestion` data structure as the value of the `Question` parameter when calling the `CreateHIT` operation. As with the `QuestionForm` data structure, an `ExternalQuestion` is a string value that consists of XML data. This data must conform to the `ExternalQuestion` schema. See Data Structure Schema Locations (p. 122) for the location of this schema. For more information about using XML data as a parameter or return value, see Using XML Parameter Values (p. 85).

> **Note**
> You can only use an external question as the question of a HIT. You cannot use an external question with a Qualification test.

The ExternalQuestion Data Structure

The `ExternalQuestion` data structure has a root element of **`ExternalQuestion`**.

The **`ExternalQuestion`** element contains the following elements:

Name	Description	Required
`ExternalURL`	The URL of your web form, to be displayed in a frame in the Worker's web browser. This URL must use the HTTPS protocol. Type: URL Default: None Amazon Mechanical Turk appends the following parameters to this URL: `assignmentId`, `hitId`, `turkSubmitTo`, and `workerId`. For more information	Yes

Name	Description	Required
	about these appended parameters, see the sections following this table.	
FrameHeight	The height of the frame, in pixels. If you set the value to 0, your HIT will automatically resize to fit within the Worker's browser window. Type: Integer Default: None	Yes

Example

The following is an example of a complete ExternalQuestion data structure. Remember that to pass this structure in as the value of a parameter to an operation, XML characters must be escaped as character entities. For more information about escaping XML characters, see Using XML Parameter Values (p. 85). See Data Structure Schema Locations (p. 122) for the location of this schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<ExternalQuestion xmlns="[the ExternalQuestion schema URL]">
  <ExternalURL>https://tictactoe.amazon.com/gamesurvey.cgi?gameid=01523</ExternalURL>
  <FrameHeight>0</FrameHeight>
</ExternalQuestion>
```

The External Form

When a Worker attempts to complete a HIT with an external question, the external website is loaded into a frame in the middle of the screen. The web page at that URL should display a form for the Worker to fill out, and all the information the Worker will need to complete the HIT.

The Frame's URL and Parameters

The URL used for the frame is the ExternalURL of the question with the following parameters appended: assignmentId, hitId, turkSubmitTo, and workerId. These parameters are appended CGI-style: The full URL has a question mark (?) before the first parameter, and an ampersand (&) between each parameter, with each parameter consisting of a name, an equal sign (=), and a value. Other parameters already present in this style in ExternalURL are preserved, so the final URL will only have one question mark, and all parameters will be separated by ampersands (&).

Note
The URL you use for the ExternalURL must use the HTTPS protocol.

For example, consider an ExternalURL of:

```
https://tictactoe.amazon.com/gamesurvey.cgi?gameid=01523
```

With this ExternalURL, the full URL used for the page in the frame could be as follows:

```
https://tictactoe.amazon.com/gamesurvey.cgi?gameid=01523
&assignmentId=123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE
&hitId=123RVWYBAZW00EXAMPLE
```

```
&turkSubmitTo=https://www.mturk.com/
&workerId=AZ3456EXAMPLE
```

Preview Mode

Your external question will be displayed when a Worker previews the HIT on the Amazon Mechanical Turk website, before the Worker has clicked the "Accept HIT" button. When the HIT is being previewed, the URL will have a special value for the `assignmentId`: `ASSIGNMENT_ID_NOT_AVAILABLE`.

When a Worker previews a HIT, your web page should show her everything she will need to do to complete the HIT, so she can decide whether or not to accept it. The easiest way to do this is to simply display the form as it would appear when the HIT is accepted. However, you may want to take precautions to prevent a Worker from accidentally filling out or submitting your form prior to accepting the HIT.

You can use JavaScript or server-side logic to check the `assignmentId` parameter, and change the display of the form if the HIT is being previewed (`assignmentId=ASSIGNMENT_ID_NOT_AVAILABLE`).

If a Worker submits your form before accepting the HIT, and your form attempts to post the data back to Amazon Mechanical Turk, Amazon Mechanical Turk will display an error message to the Worker, and the results will not be accepted.

The Form Action

The form on the external website must post the result data back to Amazon Mechanical Turk using the following URL:

```
https://www.mturk.com/mturk/externalSubmit
```

Or, if you are using the Amazon Mechanical Turk sandbox, you should post the result data back to Mechanical Turk using the following sandbox URL:

```
https://workersandbox.mturk.com/mturk/externalSubmit
```

The form must include the `assignmentId` field that was appended to the URL used to access your form. It should be submitted along with the other form fields submitted by your form, with a name of `assignmentId` and the same value as was passed to the form. Be sure to spell the field name as it appears here, with the same letters uppercase and lowercase.

> **Note**
> The field names `assignmentId`, `hitId`, `turkSubmitTo`, and `workerId` are reserved for special purposes. Your form only needs to submit the `assignmentId` field. Any data submitted with a field name of "`hitId`" will be ignored, and will not appear in the results data for the HIT.

The form must submit data to that URL using the "POST" method. The data the form submits should be name-value pairs in the CGI-style:

- Each field appears as the name, an equal sign, and the value. For example: `favoriteColor=blue`
- Data that appears in the posted URL is preceded by a question mark (`?`), and is delimited by ampersands (`&`). For example:

  ```
  https://www.mturk.com/mturk/externalSubmit?favoriteColor=blue&favoriteNumber=7&...
  ```

- Data that appears in the HTTP message body (using the "POST" method) has one data pair per line. For example:

```
favoriteColor=blue
favoriteNumber=7
...
```

The easiest way to post the data in the CGI-style is to use an HTML form on the web page, with the `externalSubmit` URL as the "action," and "POST" as the "method."

Using Crowd HTML Elements

The External Question supports Crowd HTML Elements (p. 82). Based on HTML web components, they encapsulate HTML, CSS, and JavaScript functionality behind a single HTML element. For example, the `<crowd-form>` element sets up your form for you, setting the correct submission endpoint and inserting a submit button at the end. Other elements provide question widgets or design elements you can customize to create more tailored HIT structures.

Requirements

There are three requirements for using Crowd HTML elements:

- You must place the elements between the `<crowd-form>` opening and closing elements. As noted above, these also invisibly set up a `<form>` element and place a "submit" button at the bottom.
- Place the Crowd HTML Elements loader script before the opening `<crowd-form>` element.
- Store the HTML file in an S3 bucket, make the file publicly accessible, and reference it in the `<ExternalURL>` element of your XML .

Try this sample sentiment analysis form.

Example

```
<!DOCTYPE html>
<html>
  <body>
 <script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
 <crowd-form>
   <crowd-classifier
     name="sentiment"
     categories="['Positive', 'Negative', 'Neutral', 'N/A']"
     header="What sentiment does this text convey?"
   >
     <classification-target>
     Everything is wonderful.
     </classification-target>

     <full-instructions header="Sentiment Analysis Instructions">
     <p><strong>Positive</strong>
       sentiment include: joy, excitement, delight</p>
     <p><strong>Negative</strong> sentiment include:
       anger, sarcasm, anxiety</p>
     <p><strong>Neutral</strong>: neither positive or
       negative, such as stating a fact</p>
     <p><strong>N/A</strong>: when the text cannot be
       understood</p>
     <p>When the sentiment is mixed, such as both joy and sadness,
       use your judgment to choose the stronger emotion.</p>
     </full-instructions>
```

```
    <short-instructions>
     Choose the primary sentiment that is expressed by the text.
    </short-instructions>
   </crowd-classifier>
 </crowd-form>
  </body>
</html>
```

Between the `<classification-target>` opening and closing tags, you can put any HTML that could be rendered and classified. For example you could use an audio clip or a video clip.

For more details, read the Crowd HTML Elements (p. 82) article or view the Element Reference to see the different elements that are available.

The Answer Data

When the Worker submits your form, the form sends the field data to Amazon Mechanical Turk using the `externalSubmit` URL, and Amazon Mechanical Turk records the field data as the results of the Assignment.

When you retrieve the results using `ListAssignmentsForHIT`, the field data submitted by your form will appear in the `Answer` of the Assignment (p. 124) as if each field were a free-text answer. The `QuestionIdentifier` element of the answer will be the name of the field, and the `FreeText` element will contain the value.

See the QuestionFormAnswers data format (p. 113) for more information about the format of answer data.

Guidelines For Using External Questions

External questions give your application a great deal of power over how Workers submit results for your HITs. To ensure you get good results for your HITs, you should make sure your web server and web pages can provide your Workers with a quality experience.

Because external questions depend on your web server for rendering the question form, both while Workers are previewing HITs and while Workers are completing HITs, your server will need to be engineered for high availability. The Amazon Mechanical Turk website gets heavy traffic, so your web server will need to be able to respond quickly and correctly when receiving many requests in a short period of time.

Tip

Amazon S3 offers high availability of data, accessible via public HTTPS URLs. You can host your external questions as web pages in Amazon S3, and not have to run your own server. When you use an S3-hosted layout, you need to add a `ContentType` header and set the ACL to public read, as shown in the following Python SDK PutObject call:

```
s3_client.put_object(
   ACL='public-read',
   Body=HTML layout,
   Bucket=S3 bucket name,
   Key=Object name,
   ContentType="text/html"
)
```

Your website can do many things inside the frame, but eventually it must cause the Worker's browser to load the "externalSubmit" URL in the frame with the results in POST data. The easiest way to do this is with an HTML form whose fields contain the HIT results, with a submit button that the Worker will click.

If an external HIT prevents the Worker from submitting results back to Amazon Mechanical Turk using the "externalSubmit" mechanism, the Worker may not be able to claim rewards or continue doing work without restarting their session. Amazon Mechanical Turk reserves the right to remove any external HITs that are not functioning properly.

> **Note**
> Your HIT will be rendered inside an IFRAME that has certain limitations. The IFRAME operates in HTML5 "sandbox" mode that has extra restrictions on the content that can appear in the frame. This limits your ability to execute certain code and to use technologies such as Adobe Flash. To ensure your HITs work as expected, we recommend you test them first in the Requester Sandbox.

Finally, please remember that external questions must meet the Amazon Mechanical Turk Participation Agreement, and Amazon Mechanical Turk's standards for appropriate content. Specifically, the Participation Agreement expressly prohibits the use of Amazon Mechanical Turk for advertising or solicitation. If your website typically displays advertising to visitors, please make sure those advertisements do not appear in your external questions. Amazon Mechanical Turk reserves the right to remove HITs with inappropriate content.

QuestionForm

Topics

Description

The `QuestionForm` data format describes one or more *questions* for a HIT, or for a Qualification test. It contains instructions and data Workers use to answer the questions, and a set of one or more form fields, which are rendered as a web form for a Worker to fill out and submit.

A `QuestionForm` is a string value that consists of XML data. This XML data must conform to the `QuestionForm` schema. All elements in a `QuestionForm` belong to a namespace whose name is identical to the URL of the `QuestionForm` schema document. See Schema Locations (p. 122) for the location of this schema.

> **Tip**
> For information about creating HITs that use your own web site in a frame instead of questions, see the ExternalQuestion data structure (p. 92).

The `QuestionForm` data structure is used as a parameter value for the following operations:

- `CreateHIT and CreateHITWithHITType`
- `CreateQualificationType`
- `UpdateQualificationType`

For more information about using XML data as a parameter or return value, see Using XML Parameter Values (p. 85).

QuestionForm Structure

The top-most element of the `QuestionForm` data structure is a `QuestionForm` element. This element contains optional `Overview` elements and one or more `Question` elements. There can be any number of these two element types listed in any order. The following example structure has an `Overview` element and a `Question` element followed by a second `Overview` element and `Question` element—all within the same `QuestionForm`.

```
<QuestionForm xmlns="[the QuestionForm schema URL]">
  <Overview>
    [...]
  </Overview>
  <Question>
    [...]
  </Question>
  <Overview>
    [...]
  </Overview>
  <Question>
    [...]
  </Question>
  [...]
</QuestionForm>
```

The `Overview` element describes instructions and information, and presents them separately from the set of questions. It can contain any kind of informational content, as described below. If omitted, no overview text is displayed above the questions.

Each `Question` element can contain the elements described in the following table. See also the example below the table.

Name	Description	Required
`QuestionIdentifier`	An identifier for the question. This identifier is used to associate the Worker's answers with the question in the answer data. Type: String Default: None	Yes
`DisplayName`	A name for the question, displayed as a prominent heading. Type: String Default: None	No
`IsRequired`	Specifies whether the Worker must provide an answer for this question to successfully submit the form. Type: Boolean Default: false Valid Values: true | false	No
`QuestionContent`	The instructions and data specific to this question, such as the text of the question. It can contain any kind	Yes

Name	Description	Required
	of informational content, as described in the *Content Structure* section below. Type: Content structure Default: None	
`AnswerSpecification`	A structure that describes the field type and possible values for the answer to this question, as described in the *Answer Specification* section below. This element controls how the form field is rendered and specifies which values are valid answers for this question. Type: An answer specification structure Default: None Valid Values: `FreeTextAnswer` | `SelectionAnswer` | `FileUploadAnswer`	Yes

For example:

```
<Question>
  <QuestionIdentifier>my_question_id</QuestionIdentifier>
  <DisplayName>My Question</DisplayName>
  <IsRequired>true</IsRequired>
  <QuestionContent>
    [...]
  </QuestionContent>
  <AnswerSpecification>
    [...]
  </AnswerSpecification>
</Question>
```

Content Structure

The `Overview` elements and the `QuestionContent` elements of a `QuestionForm` can contain different types of information. For example, you might include a paragraph of text and an image in your HIT's overview.

Each kind of information is defined by a corresponding element. These elements can appear in any number, in any order. The content elements are rendered in the order in which they occur in the containing element.

Following are the allowed information types:

- Title
- Text
- List
- Binary
- Application (Deprecated)
- EmbeddedBinary
- FormattedContent

Each of these types are described in detail in the following subsections. A full example showing the use of the elements and information types is at the end of the section.

Title

A `Title` element specifies a string to be rendered as a title or heading.

```
<Title>The Next Move</Title>
```

Text

A `Text` element specifies a block of text to be rendered as a paragraph. Only plain text is allowed. HTML is not allowed. If HTML characters (such as angle brackets) are included in the data, they appear verbatim in the web output.

```
<Text>What is the best next move for "X" in this game of Tic-Tac-Toe?</Text>
```

List

A `List` element displays a bulleted list of items. Items are specified using one or more `ListItem` elements inside the `List`. The `ListItem` element is a string.

```
<List>
  <ListItem>It must be a valid move.</ListItem>
  <ListItem>"X" cannot resign.</ListItem>
</List>
```

Binary

A `Binary` element specifies non-textual data of some kind, such as an image, audio, or video. The elements listed in the following table are required and must be entered in the order shown here.

Name	Description	Required
`MimeType`	Specifies the type of the data. Type: `MimeType` element Default: None Child Elements: • A required string that specifies the type of the data. The possible values are **`image`**, **`audio`**, or **`video`**. • An optional string that specifies the format of the item, such as **`gif`**	Yes
`DataURL`	The data itself specified with a `DataURL` element that contains a valid HTTP URL. Type: `DataURL` element Default: None	Yes
`AltText`	The text that should appear if the data cannot be rendered in the browser.	Yes

Name	Description	Required
	Type: String Default: None	

```
<Binary>
  <MimeType>
    <Type>image</Type>
    <SubType>gif</SubType>
  </MimeType>
  <DataURL>http://tictactoe.amazon.com/game/01523/board.gif</DataURL>
  <AltText>The game board, with "X" to move.</AltText>
</Binary>
```

Application (Deprecated)

Important
Beginning Tuesday, December 12th 2017 the QuestionForm data structure will **no longer support the `Application` element.** Instead, we recommend using the HTMLQuestion (p. 88) or ExternalQuestion (p. 92) data structures for including interactive content for Workers.

An `Application` element specifies an embedded application. It contains either a `JavaApplet` element or a `Flash` element.

You can specify zero or more parameters to pass to your Java applet or Flash application when it is opened in the web page. For a HIT, in addition to the parameters you specify, Amazon Mechanical Turk includes two parameters specific to the HIT: `hitId` and `assignmentId`. The `hitId` parameter is equal to the ID of the HIT. The `assignmentId` parameter is equal to the ID of the assignment if the Worker has accepted the HIT, or equal to `ASSIGNMENT_ID_NOT_AVAILABLE` if the Worker is only previewing the HIT.

The `JavaApplet` element includes the elements described in the following table:

Name	Description	Required
`AppletPath`	The URL path to the directory that contains Java classes for the applet. Type: URL Default: None	Yes
`AppletFilename`	The name of the class file that contains the applet code, which is located in the path specified by `AppletPath`. Type: String Default: None	Yes
`Width`	The width of the bounding box for the applet. Type: String Default: None	Yes

Name	Description	Required
`Height`	The height of the bounding box for the applet. Type: String Default: None	Yes
`ApplicationParameter`	The parameters for the applet. Type: `ApplicationParameter` Default: None Child Elements: • A required string that specifies the name of the parameter • A required string that specifies the value of the parameter	No

The `Flash` element includes the elements described in the following table:

Name	Description	Required
`FlashMovieURL`	The URL of the Flash movie file. Type: URL Default: None	Yes
`Width`	The width of the bounding box for the Flash movie. Type: String Default: None	Yes
`Height`	The height of the bounding box for the Flash movie. Type: String Default: None	Yes
`ApplicationParameter`	The parameters for the Flash movie. Type: `ApplicationParameter` Default: None Child Elements: • A required string that specifies the name of the parameter • A required string that specifies the value of the parameter	No

```
<Application>
```

```
  <JavaApplet>
    <AppletPath>http://tictactoe.amazon.com/applets/</AppletPath>
    <AppletFilename>GameViewer.class</AppletFilename>
    <Width>400</Width>
    <Height>300</Height>
    <ApplicationParameter>
      <Name>game_id<Name>
      <Value>01523</Value>
    </ApplicationParameter>
  </JavaApplet>
</Application>
```

EmbeddedBinary

An `EmbeddedBinary` element specifies an external object of non-textual data of some kind, such as an image, audio or video, that displays in your browser. The elements listed in the following table are required and must be entered in the order shown here.

Name	Description	Required
`EmbeddedMimeType`	Specifies the type of the data. Type: `EmbeddedMimeType` element Default: None Child Elements: • A required string that specifies the type of the data. The possible values are **`image`**, **`audio`**, or **`video`**. • An optional string that specifies the format of the item, such as **`gif`**	Yes
`DataURL`	The data itself specified by a `DataURL` element that contains a valid HTTP URL Type: `DataURL` element Default: None	Yes
`AltText`	The text that should appear if the data cannot be rendered in the browser. Type: String Default: None	Yes
`Width`	The width of the bounding box for the object. Type: String Default: None	Yes
`Height`	The height of the bounding box for the object. Type: String Default: None	Yes
`ApplicationParameter`	The parameters for the `EmbeddedBinary` object.	No

Name	Description	Required
	Type: `ApplicationParameter` Default: None Child elements: • A required string that specifies the name of the parameter • A required string that specifies the value of the parameter	

```
<EmbeddedBinary>
  <EmbeddedMimeType>
    <Type>image</Type>
    <SubType>gif</SubType>
  </EmbeddedMimeType>
  <DataURL>http://tictactoe.amazon.com/game/01523/board.gif</DataURL>
  <AltText>The game board, with "X" to move.</AltText>
  <Width>400</Width>
  <Height>300</Height>
  <ApplicationParameter>
    <Name>game_id<Name>
    <Value>01523</Value>
  </ApplicationParameter>
</EmbeddedBinary>
```

FormattedContent

For finer control over the display of your HIT information, you can specify a `FormattedContent` element. Formatted content is a block of text with formatting information specified using XHTML tags. For example, you can use XHTML tags to specify that certain words appear in a boldface font or to include a table in your HIT information.

Only a limited subset of XHTML is supported. For more information on the creating and validating XHTML formatted content, see Formatted Content: XHTML (p. 115).

The value of the `FormattedContent` element must be specified as an XML CDATA block. CDATA tells the web service that the XHTML elements are not part of the `QuestionForm` data schema. For example, the following describes a paragraph of formatted text:

```
<FormattedContent><![CDATA[
  <p>This is a paragraph with <b>bold text</b>,
  <i>italic text</i>, and <b><i>bold italic text</i></b>.</p>
]]></FormattedContent>
```

Answer Specification

The `AnswerSpecification` element describes the format and possible values for answers to a question. It contains a `FreeTextAnswer` element, which describes a text field; a `SelectionAnswer` element, which describes a multiple choice field; or a `FileUploadAnswer`, which prompts the Worker to upload a file as the answer to the question.

FreeTextAnswer

A `FreeTextAnswer` element describes a text field and constraints on its possible values. It includes the elements described in the following table:

Name	Description	Required
`Constraints`	Describes the constraints on the allowed values for the text field. This element is described in the next table. Type: `Constraints` element Default: None	No
`DefaultText`	Specifies default text. This value appears in the form when it is rendered, and is accepted as the answer if the Worker does not change it. Type: String Default: An empty value	No
`NumberOfLinesSuggestion`	Specifies how tall the form field should be, if possible. The field might be rendered as a text box with this many lines, depending on the device the Worker is using to see the form. Type: Integer Default: 1	No

Note
A Qualification test that is to be graded automatically using an answer key cannot have any free-text questions. An answer key can only match multiple-choice questions and cannot match free-text fields.

The optional `Constraints` element describes constraints on the allowed values for the text field. If no constraints are specified, any value is accepted for the field.

The `Constraints` element includes the elements described in the following table:

Name	Description	Required
`IsNumeric`	Specifies that the value entered must be numeric. Type: empty element Default: None Attributes: • An optional integer that specifies the minimum value allowed • An optional integer that specifies the maximum value allowed	No
`Length`	Specifies the length range of the answer.	No

Name	Description	Required
	Type: empty element Default: None Attributes: • An optional non-negative integer that specifies the minimum number of characters • An optional positive integer that specifies the maximum number of characters	
`AnswerFormatRegex`	Specifies that JavaScript validates the answer string against a given pattern. **Note** A limitation of this approach is that Workers who have disabled JavaScript on their browsers cannot validate their answers. Although this is uncommon, you might want to caution your Workers. Type: empty element Default: None Attributes: • A required string that specifies the regular expression that JavaScript uses to validate against the Workers' entered values • An optional string that allows you to edit the content of errors displayed to the Worker on the Worker web site if the regex validation fails. If this attribute is not specified, the error displayed is "Invalid input supplied." • An optional string with the value `i` which specifies that case is ignored when matching characters	No

The `Constraints` element can contain multiple `AnswerFormatRegex` elements. All `AnswerFormatRegex` constraints must be satisfied before the Worker can submit the HIT.

The following examples demonstrate how to use the `FreeTextAnswer` element.

If you want a 3-digit positive integer between 100 and 999, use the following:

```
<FreeTextAnswer>
  <Constraints>
    <IsNumeric minValue="100" maxValue="999"/>
    <Length minLength="3" maxLength="3"/>
  </Constraints>
</FreeTextAnswer>
```

If you want a 3-digit number that includes decimals, use the following:

```
<FreeTextAnswer>
```

```
  <Constraints>
     <IsNumeric/>
     <Length minLength="3" maxLength="3"/>
  </Constraints>
</FreeTextAnswer>
```

If you want to ensure that there is some text, use the following example. The **`minLength`** attribute includes whitespaces in the character count.

```
<FreeTextAnswer>
  <Constraints>
    <Length minLength="2" />
    <AnswerFormatRegex regex="\S" errorText="The content cannot be blank."/>
  </Constraints>
</FreeTextAnswer>
```

If you specify the **`minLength`** attribute, it is the same as if the `IsRequired` element is `true`. If you want to allow an *optional* string that must be at least two characters, use the following:

```
<FreeTextAnswer>
  <Constraints>
    <AnswerFormatRegex regex="(^$|\S{2,})"
     errorText="You must enter at least two characters."/>
  </Constraints>
</FreeTextAnswer>
```

To request a US phone number in the format 1-nnn-nnn-nnnn, where "1-" is optional, use the following:

```
<FreeTextAnswer>
  <Constraints>
    <AnswerFormatRegex
     regex="^(1[- ]?)?(\([2-9]\d{2}\)\s*|[2-9]\d{2}-?)[2-9]\d{2}-?\d{4}$)"
     errorText="You must enter a US phone number in the format
     1-555-555-1234 or 555-555-1234."/>
  </Constraints>
</FreeTextAnswer>
```

If you want an answer that contains a date formatted as yyyy-mm-dd, use the following:

```
<FreeTextAnswer>
  <Constraints>
    <AnswerFormatRegex regex="^[12][0-9]{3}-[01]?\d-[0-3]?\d$"
     errorText="You must enter a date with the format yyyy-mm-dd."/>
  </Constraints>
</FreeTextAnswer>
```

If you want an answer that contains "regex" and variations including RegEx, REGex, and RegExes, use the following:

```
<FreeTextAnswer>
```

```
  <Constraints>
    <AnswerFormatRegex regex="regex" flags="i"
     errorText="You must enter 'regex'."/>
  </Constraints>
</FreeTextAnswer>
```

SelectionAnswer

A `SelectionAnswer` describes a multiple-choice question. Depending on the element defined, the Worker might be able to select zero, one, or multiple items from a set list as the answer to the question.

A `SelectionAnswer` element includes the elements described in the following table:

Name	Description	Required
`MinSelectionCount`	Specifies the minimum number of selections allowed for a valid answer. This value can range from 0 to the number of selections. Type: non-negative Integer Default: 1	No
`MaxSelectionCount`	Specifies the maximum number of selections allowed for a valid answer. This value can range from 1 to the number of selections. Type: positive Integer Default: 1	No
`StyleSuggestion`	Specifies what style of multiple-choice form field to use when displaying the question to the Worker. The field might not use the suggested style, depending on the device the Worker is using to see the form. Type: String Default: None Valid Values: • Can be used if `MaxSelectionCount` is **1**, because it restricts the user to selecting either zero or one item from the list • Allows multiple selections, but can be restricted by using the `MaxSelectionCount` element • Allows multiple selections, but can be restricted by using the `MaxSelectionCount` element • Can be used if `MaxSelectionCount` is **1**, because it restricts the user to selecting either zero or one item from the list • Allows multiple selections, but can be restricted by using the `MaxSelectionCount` element • Allows multiple selections, but can be restricted by using the `MaxSelectionCount` element	No

Name	Description	Required
`Selections`	Specifies the answer selections. Type: `Selections` structure Default: None Child elements: • Specifies an answer selection. This element is described fully in the next table. • An optional text field to display below the selection list that allows the Worker to enter an alternate answer that does not appear in the list of selections. The contents of this element are similar to `FreeTextAnswer`. **Note** A Qualification test that you want to grade automatically using an answer key cannot have an `OtherSelection` field for a multiple choice question. An answer key can only match multiple-choice questions and cannot match free-text fields.	Yes

The `Selections` element lists the selection options available. It contains one or more `Selection` elements, one for each possible answer in the set. The `Selection` element includes the elements described in the following table:

Name	Description	Required
`SelectionIdentifier`	A unique alphanumeric string that is in the answer data if this selection is chosen. Type: String Default: None	Yes
One of the following elements:		Yes
`Text`	Contains the content of the selected item. Type: String Default: None	
`FormattedContent`	A block of text formatted using XHTML tags that contains the content of the selected item. For more information about this format, see Formatted Content: XHTML (p. 115). Type: String Default: None	
`Binary`	Contains the content of the selected item.	

Name	Description	Required
	Type: `Binary` Default: None	

The following example shows a `SelectionAnswer` element that specifies a question with four radiobuttons.

```
<SelectionAnswer>
  <StyleSuggestion>radiobutton</StyleSuggestion>
  <Selections>
    <Selection>
      <SelectionIdentifier>C1</SelectionIdentifier>
      <Text>C1 (northeast)</Text>
    </Selection>
    <Selection>
      <SelectionIdentifier>C2</SelectionIdentifier>
      <Text>C2 (east)</Text>
    </Selection>
    <Selection>
      <SelectionIdentifier>A3</SelectionIdentifier>
      <Text>A3 (southwest)</Text>
    </Selection>
    <Selection>
      <SelectionIdentifier>C3</SelectionIdentifier>
      <Text>C3 (southeast)</Text>
    </Selection>
  </Selections>
</SelectionAnswer>
```

FileUploadAnswer (Deprecated)

Important
Beginning Tuesday, December 12th 2017 the Answer Specification structure will **no longer support the `FileUploadAnswer` element** to be used for the QuestionForm data structure. Instead, we recommend that Requesters who want to create HITs asking Workers to upload files use Amazon S3.

A `FileUploadAnswer` prompts the Worker to upload a file as the answer to the question. When the Worker uploads the file, Amazon Mechanical Turk stores the file separately from the answer data. Once the HIT is submitted, your application can call the `GetFileUploadURL` operation to get a temporary URL it can use to download the file.

The `FileUploadAnswer` specification contains two required elements, a `MinFileSizeInBytes` and a `MaxFileSizeInBytes`, that specify the minimum and maximum allowed file sizes respectively. If the Worker uploads a file whose size in bytes is outside of this range, the answer is rejected, and the Worker must upload a different file to complete the HIT. You can specify a maximum size up to 2000000000 (2 billion) bytes.

Note
A `FileUploadAnswer` element can only be used with HITs. It cannot be used with Qualification tests.

The following example demonstrates a `FileUploadAnswer` element that specifies a file with a minimum of 1000 bytes and a maximum of 3000000 bytes.

```
<FileUploadAnswer>
  <MaxFileSizeInBytes>3000000</MaxFileSizeInBytes>
  <MinFileSizeInBytes>1000</MinFileSizeInBytes>
```

```
</FileUploadAnswer>
```

Example

The following is an example of a complete `QuestionForm` data structure. Remember that to pass this structure in as a value of a parameter to an operation, XML characters must be escaped as character entities. (See Using XML Parameter Values (p. 85) for more information.)

```
<QuestionForm xmlns="[the QuestionForm schema URL]">
  <Overview>
    <Title>Game 01523, "X" to play</Title>
    <Text>
      You are helping to decide the next move in a game of Tic-Tac-Toe.  The board looks
 like this:
    </Text>
    <Binary>
      <MimeType>
        <Type>image</Type>
        <SubType>gif</SubType>
      </MimeType>
      <DataURL>http://tictactoe.amazon.com/game/01523/board.gif</DataURL>
      <AltText>The game board, with "X" to move.</AltText>
    </Binary>
    <Text>
      Player "X" has the next move.
    </Text>
  </Overview>
  <Question>
    <QuestionIdentifier>nextmove</QuestionIdentifier>
    <DisplayName>The Next Move</DisplayName>
    <IsRequired>true</IsRequired>
    <QuestionContent>
      <Text>
        What are the coordinates of the best move for player "X" in this game?
      </Text>
    </QuestionContent>
    <AnswerSpecification>
      <FreeTextAnswer>
        <Constraints>
          <Length minLength="2" maxLength="2" />
        </Constraints>
        <DefaultText>C1</DefaultText>
      </FreeTextAnswer>
    </AnswerSpecification>
  </Question>
  <Question>
    <QuestionIdentifier>likelytowin</QuestionIdentifier>
    <DisplayName>The Next Move</DisplayName>
    <IsRequired>true</IsRequired>
    <QuestionContent>
      <Text>
        How likely is it that player "X" will win this game?
      </Text>
    </QuestionContent>
    <AnswerSpecification>
      <SelectionAnswer>
        <StyleSuggestion>radiobutton</StyleSuggestion>
        <Selections>
          <Selection>
            <SelectionIdentifier>notlikely</SelectionIdentifier>
            <Text>Not likely</Text>
          </Selection>
          <Selection>
```

```
            <SelectionIdentifier>unsure</SelectionIdentifier>
            <Text>It could go either way</Text>
          </Selection>
          <Selection>
            <SelectionIdentifier>likely</SelectionIdentifier>
            <Text>Likely</Text>
          </Selection>
        </Selections>
      </SelectionAnswer>
    </AnswerSpecification>
  </Question>
</QuestionForm>
```

QuestionFormAnswers

Topics

Description

The `QuestionFormAnswers` data format describes answers submitted by a Worker for a HIT, or for a Qualification test.

A `QuestionFormAnswers` data structure is a string value that consists of XML data. The XML data must conform to the QuestionForm schema. See Schema Locations (p. 122) for the location of this schema. For more information about using XML data as parameter or return value, see Using XML Parameter Values (p. 85).

> **Note**
> Answer data is *not* guaranteed by the Amazon Mechanical Turk Service to conform to the answer specifications described in a QuestionForm. MTS only guarantees that answer data returned by the service will conform to the `QuestionFormAnswers` schema. Your application should check that the answer data sufficiently answers the question.

The `QuestionFormAnswers` data structure is used as a response element for the following operations:

- `GetAssignmentsForHIT`
- `GetQualificationRequests`

All elements in a `QuestionFormAnswers` belong to a namespace whose name is identical to the URL of the `QuestionFormAnswers` schema document for the version of the API you are using.

The Structure of Answers

A `QuestionFormAnswers` element contains an `Answer` element for each question in the HIT or Qualification test for which the Worker provided an answer. Each `Answer` contains a `QuestionIdentifier` element whose value corresponds to the `QuestionIdentifier` of a `Question` in the QuestionForm. See the QuestionForm data structure (p. 97) for more information about questions and answer specifications.

If the question expects a free-text answer, the `Answer` element contains a `FreeText` element. This element contains the Worker's answer.

If the question expects a multiple-choice answer, the `Answer` element contains a `SelectionIdentifier` element for each option the Worker selected. If the Worker did not make any selections, the `Answer` will contain zero `SelectionIdentifier` elements. The identifier corresponds to the `SelectionIdentifier` for the selection provided in the answer specification for the question.

If the multiple-choice question includes an `OtherSelection` field, and the Worker enters data into this field, that data appears in the `Answer` in an `OtherSelectionText` element. If the Worker both selects an option from the list and provides text in this field, both values will be present in the answer.

If the question expects an uploaded file as an answer, the `Answer` element contains an `UploadedFileSizeInBytes` element, and an `UploadedFileKey` element. `UploadedFileSizeInBytes` indicates the size of the file the Worker uploaded. `UploadedFileKey`

is a unique identifier for the file, unique with respect to other files that Workers may have uploaded. To retrieve an uploaded file, your application calls the `GetFileUploadURL` operation, which returns a temporary URL your application can use to download the file. See the `GetFileUploadURL` operation for more information on retrieving uploaded files.

Answer data will always conform to the answer specification provided in the HIT question, or in the Qualification test question.

Example

The following is an example of a complete `QuestionFormAnswers` data structure. Remember that this value will be returned as a single return value, XML escaped in the response.

```
<QuestionFormAnswers xmlns="[the QuestionFormAnswers schema URL]">
  <Answer>
    <QuestionIdentifier>nextmove</QuestionIdentifier>
    <FreeText>C3</FreeText>
  </Answer>
  <Answer>
    <QuestionIdentifier>likelytowin</QuestionIdentifier>
    <SelectionIdentifier>notlikely</SelectionIdentifier>
  </Answer>
</QuestionFormAnswers>
```

When using Crowd HTML Elements (p. 82) in your form, they will output JSON formatted data in the `<FreeText>` field. For example, the output for a crowd-bounding-box with three boxes on the image would contain a text string similar to the one below.

```
<FreeText>
  {
    'annotatedResult':
    {
      'boundingBoxes':
      [
        {
          'height': 902,
          'label': 'human',
          'left': 53,
          'top': 174,
          'width': 619
        },
        {
          'height': 936,
          'label': 'human',
          'left': 734,
          'top': 73,
          'width': 684
        },
        {
          'height': 686,
          'label': 'human',
          'left': 1174,
          'top': 121,
          'width': 556
        }
      ],
      'inputImageProperties':
      {
        'height': 1080,
        'width': 1920
      }
    }
```

```
  }
</FreeText>
```

Formatted Content: XHTML

Topics

When you create a HIT or a Qualification test, you can include various kinds of content to be displayed to the Worker on the Amazon Mechanical Turk web site, such as text (titles, paragraphs, lists), media (pictures, audio, video) and browser applets (Java or Flash).

You can also include blocks of formatted content. Formatted content lets you include XHTML tags directly in your instructions and your questions for detailed control over the appearance and layout of your data.

You include a block of formatted content by specifying a `FormattedContent` element in the appropriate place in your QuestionForm data structure (p. 97). You can specify any number of `FormattedContent` elements in content, and you can mix them with other kinds of content.

The following example uses other content types (`Title`, `Text`) along with `FormattedContent` to include a table in a HIT:

```
<Text>
  This HIT asks you some questions about a game of Tic-Tac-Toe
  currently in progress.  Your answers will help decide the next move.
</Text>
<Title>The Current Board</Title>
<Text>
  The following table shows the board as it currently stands.
</Text>
<FormattedContent><![CDATA[
<table border="1">
  <tr>
    <td></td>
    <td align="center">1</td>
    <td align="center">2</td>
    <td align="center">3</td>
  </tr>
  <tr>
    <td align="right">A</td>
    <td align="center"><b>X</b></td>
    <td align="center"> </td>
    <td align="center"><b>O</b></td>
  </tr>
  <tr>
    <td align="right">B</td>
    <td align="center"> </td>
    <td align="center"><b>O</b></td>
    <td align="center"> </td>
  </tr>
  <tr>
    <td align="right">C</td>
    <td align="center"> </td>
    <td align="center"> </td>
    <td align="center"><b>X</b></td>
```

```
  </tr>
  <tr>
    <td align="center" colspan="4">It is <b>X</b>'s turn.</td>
  </tr>
</table>
]]></FormattedContent>
```

For more information about describing the contents of a HIT or Qualification test, see the QuestionForm data structure (p. 97).

Using Formatted Content

As you can see in the example above, formatted content is specified in an XML CDATA block, inside a `FormattedContent` element. The CDATA block contains the text and XHTML markup to display in the Worker's browser.

Only a subset of the XHTML standard is supported. For a complete list of supported XHTML elements and attributes, see the table below. In particular, JavaScript, element IDs, `class` and `style` attributes, and `<div>` and `<span>` elements are not allowed.

XML comments (`<!-- ... -->`) are not allowed in formatted content blocks.

Every XHTML tag in the CDATA block must be closed before the end of the block. For example, if you start an XHTML paragraph with a `<p>` tag, you must end it with a `</p>` tag within the same `FormattedContent` block.

> **Note**
> The tag closure requirement means you cannot open an XHTML tag in one `FormattedContent` block and close it in another. There is no way to "wrap" other kinds of question form content in XHTML. `FormattedContent` blocks must be self-contained.

XHTML tags must be nested properly. When tags are used inside other tags, the inner-most tags must be closed before outer tags are closed. For example, to specify that some text should appear in bold italics, you would use the `<b>` and `<i>` tags as follows:

```
<b><i>This text appears bold italic.</i></b>
```

But the following would not be valid, because the closing `</b>` tag appears before the closing `</i>` tag:

```
<b><i>These tags don't nest properly!</b></i>
```

Finally, formatted content must meet other requirements to validate against the XHTML schema. For instance, tag names and attribute names must be all lowercase letters, and attribute values must be surrounded by quotes.

For details on how Amazon Mechanical Turk validates XHTML formatted content blocks, see "How XHTML Formatted Content Is Validated," below.

Supported XHTML Tags

`FormattedContent` supports a limited subset of the XHTML 1.0 ("transitional") standard. The complete list of supported tags and attributes appears in the table below. Notable differences with the standard include:

- JavaScript is not allowed. The `<script>` tag is not supported, and anchors (`<a>`) and images (`<img>`) cannot use `javascript:` targets in URLs.

- CSS is not allowed. The `<style>` tag is not supported, and the `class` and `style` attributes are not supported. The `id` attribute is also not supported.
- XML comments (`<!-- ... -->`) are not supported.
- URL methods in anchor targets and image locations are limited to the following: `http:// https:// ftp:// news:// nntp:// mailto:// gopher:// telnet://`

Other things to note with regards to supported tags and attributes:

- In addition to the attributes listed, the `title` attribute is supported for all tags, and the `dir` and `lang` attributes are supported for all tags except `<br>`.
- The `alt` attribute is required for `<area>` and `<img>` tags.
- `<img>` tags also require a `src` attribute.
- `<map>` tags require a `name` attribute.

The following table lists the supported tags and attributes:

Tag	Attributes
a	accesskey charset coords href hreflang name rel rev shape tabindex target type
area	alt coords href nohref shape target
b	
big	
blockquote	cite
br	
center	
cite	
code	
col	align char charoff span valign width
colgroup	align char charoff span valign width
dd	
del	cite datetime
dl	
em	
font	color face size
h1	align
h2	align
h3	align
h4	align

Tag	Attributes
h5	align
h6	align
hr	align noshade size width
i	
img	align alt border height hspace ismap longdesc src usemap vspace width
ins	cite datetime
li	type value
map	name
ol	compact start type
p	align
pre	width
q	cite
small	
strong	
sub	
sup	
table	align bgcolor border cellpadding cellspacing frame rules summary width
tbody	align char charoff valign
td	abbr align axis bgcolor char charoff colspan headers height nowrap rowspan scope valign width
tfoot	align char charoff valign
th	abbr align axis bgcolor char charoff colspan headers height nowrap rowspan scope valign width
thead	align char charoff valign
tr	align bgcolor char charoff valign
u	
ul	compact type

How XHTML Formatted Content Is Validated

When you create a HIT or a Qualification test whose content uses `FormattedContent`, Amazon Mechanical Turk attempts to validate the formatted content blocks against a schema. If the formatted content does not validate against the schema, the operation call will fail and return an error.

To validate the formatted content, Amazon Mechanical Turk takes the contents of the `FormattedContent` element (the text and markup inside the CDATA), then constructs an XML document with an appropriate XML header, `<FormattedContent>` as the root element, and the text and markup as the element's contents (without the CDATA). This document is then validated against a schema.

For example, consider the following `FormattedContent` block:

```
...
<FormattedContent><![CDATA[
  I absolutely <i>love</i> chocolate ice cream!
]]></FormattedContent>
...
```

To validate this block, Amazon Mechanical Turk produces the following XML document:

```
<?xml version="1.0"?>
<FormattedContent xmlns="http://www.w3.org/1999/xhtml">
  I absolutely <i>love</i> chocolate ice cream!
</FormattedContent>
```

The schema used for validation is called `FormattedContentXHTMLSubset.xsd`. For information on how to download this schema, see Data Structure Schema Locations (p. 122).

You do not need to specify the namespace of the XHTML tags in your formatted content. This is assumed automatically during validation.

AnswerKey

Topics

Description

The AnswerKey data structure specifies answers for a Qualification test, and a mechanism to use to calculate a score from the key and a Worker's answers.

An AnswerKey data structure is a string value that consists of XML data. The XML data must conform to the AnswerKey schema. See WSDL and Schema Locations (p. 122) for the location of this schema. For more information about using XML data as parameter or return value, see Using XML Parameter Values (p. 85).

The AnswerKey data structure is used as a parameter for the following operations:

- `CreateQualificationType`

The AnswerKey data structure is used as a return value for the following operations:

- `GetQualificationType`

All elements in a `AnswerKey` belong to a namespace whose name is identical to the URL of the `AnswerKey` schema document for the version of the API you are using.

The Structure of an Answer Key

An answer key is contained in a `AnswerKey` element. This element contains a `Question` element for each question in the Qualification test, and an optional `QualificationValueMapping` element that describes how to calculate the Qualification value from the answer key and the Worker's answers.

Question

A `Question` element contains a `QuestionIdentifier` element, which identifies the question for this answer. This value corresponds to a `QuestionIdentifier` in the QuestionForm.

A `Question` element has one or more `AnswerOption` elements, one for each combination of selections in the multiple-choice question that affects the Worker's test score.

Each `AnswerOption` contains one or more `SelectionIdentifier` elements that correspond to identifiers for the selections in the QuestionForm. It also contains an `AnswerScore` element, a number that is added to the Worker's test score if the Worker's answer matches this option. The Worker must select all of the selections specified by the `SelectionIdentifier` elements, and no others, to earn the score.

Tip
An `AnswerScore` for an `AnswerOption` may be negative.

The `Question` may have an optional `DefaultScore`, a number that is added to the Worker's test score if none of the answer options exactly match the Worker's answer for the question. `DefaultScore` is optional, and defaults to 0.

```
<AnswerOption>
  <SelectionIdentifier>apples</SelectionIdentifier>
  <AnswerScore>10</AnswerScore>
</AnswerOption>
```

QualificationValueMapping

The `Question` may have an optional `QualificationValueMapping` element that describes how to calculate the Worker's overall score from the scores of the Worker's answers. It contains either a `PercentageMapping` element, a `ScaleMapping` element, or a `RangeMapping` element.

If no `QualificationValueMapping` is specified, the sum of the scores of the answers is used as the Qualification value.

```
<QualificationValueMapping>
  ...
</QualificationValueMapping>
```

A `PercentageMapping` specifies a maximum score for the test, as a `MaximumSummedScore` element. The Qualification value is calculated as the sum of the scores of the selected answers, divided by the maximum, multiplied by 100 and rounded to the nearest integer to produce a percentage.

```
  ...
  <PercentageMapping>
    <MaximumSummedScore>15</MaximumSummedScore>
  </PercentageMapping>
```

A `ScaleMapping` specifies a multiplier, as a decimal value in a `SummedScoreMultiplier` element. The Qualification value is calculated as the sum of the scores of the selected answers, multiplied by the multiplier.

```
...
<ScaleMapping>
  <SummedScoreMultiplier>3</SummedScoreMultiplier>
</ScaleMapping>
```

A `RangeMapping` assigns specific Qualification values to ranges of total test scores. It contains one or more `SummedScoreRange` elements, each of which specify an `InclusiveLowerBound` element, an `InclusiveUpperBound` element, and a `QualificationValue` that becomes the Qualification value if the sum of the scores of the selected answers falls within the specified range. Finally, the `RangeMapping` includes a single `OutOfRangeQualificationValue`, which specifies the Qualification value if the sum of the scores of the selected answers do not fall within a specified range.

Note
Ranges cannot overlap. If ranges overlap, the behavior is undefined.

```
...
<RangeMapping>
  <SummedScoreRange>
    <InclusiveLowerBound>5</InclusiveLowerBound>
    <InclusiveUpperBound>7</InclusiveUpperBound>
    <QualificationValue>5</QualificationValue>
  </SummedScoreRange>
  <SummedScoreRange>
    <InclusiveLowerBound>8</InclusiveLowerBound>
    <InclusiveUpperBound>10</InclusiveUpperBound>
    <QualificationValue>10</QualificationValue>
  </SummedScoreRange>
  <OutOfRangeQualificationValue>0</OutOfRangeQualificationValue>
</RangeMapping>
```

Example

The following is an example of a complete `AnswerKey` data structure. Remember that to pass this structure in as a parameter to an operation, XML characters must be escaped as character entities. For more information, see Using XML Parameter Values (p. 85).

```
<AnswerKey xmlns="[the AnswerKey schema URL]">
  <Question>
    <QuestionIdentifier>nextmove</QuestionIdentifier>
    <AnswerOption>
      <SelectionIdentifier>D</SelectionIdentifier>
      <AnswerScore>5</AnswerScore>
    </AnswerOption>
  </Question>
  <Question>
    <QuestionIdentifier>favoritefruit</QuestionIdentifier>
    <AnswerOption>
      <SelectionIdentifier>apples</SelectionIdentifier>
      <AnswerScore>10</AnswerScore>
    </AnswerOption>
  </Question>
  <QualificationValueMapping>
    <PercentageMapping>
      <MaximumSummedScore>15</MaximumSummedScore>
    </PercentageMapping>
  </QualificationValueMapping>
</AnswerKey>
```

Data Structure Schema Locations

The Amazon Mechanical Turk uses several XML data structures to help you define your tasks flexibly. These data structures are specified using schemas that are versioned. This allows MTurk to add new versions of task types while preserving backwards compatibility.

When constructing an XML object for any of these structures, you must declare a namespace that matches the target namespace for the schema for the structure. The namespace is defined using the URL to the schema definition. For example, here is how to declare your namespace when constructing an HTMLQuestion:

```
<HTMLQuestion
xmlns="http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2011-11-11/
HTMLQuestion.xsd">
[...]
</HTMLQuestion>
```

If the service returns an error message about data not validating against the schema, make sure your namespace declaration matches the target namespace specified in the schema.

> **Important**
> Beginning Tuesday, December 12th 2017 the QuestionForm (p. 97) structure will **no longer support the `FileUploadAnswer` element and the `Application` element.** The **2017-11-06** version of the QuestionForm schema has been updated to reflect these changes. If you don't use the deprecated elements in your QuestionForm, the **2005-10-01** schema will continue to work.

You can find the schema namespace values for all of the question and answer data structures below:

Type of Schema	Latest Version	Schema Namespace
The HTMLQuestion (p. 88) schema	`2011-11-11`	http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2011-11-11/HTMLQuestion.xsd
The ExternalQuestion (p. 92) schema	`2006-07-14`	http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2006-07-14/ExternalQuestion.xsd
The Formatted Content: XHTML (p. 115) subset	`2006-07-14`	http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2006-07-14/FormattedContentXHTMLSubset.xsd
The QuestionForm (p. 97) schemas	`2017-11-06`	http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2017-11-06/QuestionForm.xsd
The QuestionFormAnswers (p. 113) schemas	`2005-10-01`	http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2005-10-01/QuestionFormAnswers.xsd
The AnswerKey (p. 119) schemas schemas	`2005-10-01`	http://mechanicalturk.amazonaws.com/

Type of Schema	Latest Version	Schema Namespace
		AWSMechanicalTurkDataSchemas/2005-10-01/AnswerKey.xsd

Data Structures

Topics

The Amazon Mechanical Turk API uses several common data structures in its operation request and response structures. For easy reference, these structures are documented in separate articles. For more information, refer to their corresponding operations.

Assignment

Description

The Assignment data structure represents a single assignment of a HIT to a Worker. The assignment tracks the Worker's efforts to complete the HIT, and contains the results for later retrieval.

The Assignment data structure is used as a response element for the following operations:

- `GetAssignment`
- `ListAssignmentsForHIT`

Elements

The Assignment structure can contain the following elements.

Name	Description	Required
`AssignmentId`	A unique identifier for the assignment. Can be up to 255 bytes in length. Type: String Default: None	No
`WorkerId`	The ID of the Worker who accepted the HIT. Can be up to 255 bytes in length. Type: String Default: None	No

Name	Description	Required
`HITId`	The ID of the HIT. Can be up to 255 bytes in length. Type: String Default: None	No
`AssignmentStatus`	The status of the assignment Type: String Valid Values: Submitted | Approved | Rejected Default: None	No
`AutoApprovalTime`	If results have been submitted, `AutoApprovalTime` is the date and time the results of the assignment results are considered **Approved** automatically if they have not already been explicitly approved or rejected by the Requester. This value is derived from the auto-approval delay specified by the Requester in the HIT. This value is omitted from the assignment if the Worker has not yet submitted results. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None	No
`AcceptTime`	The date and time the Worker accepted the assignment. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None	No
`SubmitTime`	If the Worker has submitted results, `SubmitTime` is the date and time the assignment was submitted. This value is omitted from the assignment if the Worker has not yet submitted results. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None	No

Name	Description	Required
`ApprovalTime`	If the Worker has submitted results and the Requester has approved the results, `ApprovalTime` is the date and time the Requester approved the results. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None. **Note**: This value is omitted from the assignment if the Requester has not yet approved the results.	No
`RejectionTime`	If the Worker has submitted results and the Requester has rejected the results, `RejectionTime` is the date and time the Requester rejected the results. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None. **Note**: This value is omitted from the assignment if the Requester has not yet rejected the results.	No
`Deadline`	The date and time of the deadline for the assignment. This value is derived from the deadline specification for the HIT and the date and time the Worker accepted the HIT. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None	No
`Answer`	The Worker's answers submitted for the HIT contained in a `QuestionFormAnswers` document, if the Worker provides an answer. If the Worker does not provide any answers, `Answer` may contain a `QuestionFormAnswers` document, or `Answer` may be empty. Type: a QuestionFormAnswers (p. 113) data structure Default: None	No
`RequesterFeedback`	The feedback string included with the call to the `ApproveAssignment` operation or the `RejectAssignment` operation, if the Requester approved or rejected the assignment and specified feedback. Type: String Default: None	No

Example

The following example shows an Assignment data structure returned by the `ListAssignmentsForHIT` operation. The `ListAssignmentsForHIT` operation returns zero or more `Assignment` elements for a **Reviewable** HIT.

```
Assignment:{
 AssignmentId: "123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE",
 WorkerId:"AZ3456EXAMPLE",
 HITId:"123RVWYBAZW00EXAMPLE",
 AssignmentStatus:"Submitted",
 Deadline: "2005-12-01T23:59:59Z",
 AcceptTime: "2005-12-01T12:00:00Z",
 SubmitTime: "2005-12-07T23:59:59Z",
 Answer:{
  QuestionFormAnswers:[XML-encoded Answer data]
  }
 }
  }
```

HIT

Description

The HIT data structure represents a single HIT, including all the information necessary for a Worker to accept and complete the HIT.

The HIT data structure is used as a response element for the following operations:

- `CreateHIT`
- `GetHIT`
- `ListReviewableHITs`
- `ListHITs`

Elements

The HIT structure can contain the elements described in the following table.

Name	Description	Required
`HITId`	A unique identifier for the HIT. The `CreateHIT` operation gives a HIT the HIT ID and the HIT retains that ID forever. Type: String Default: None	No
`HITTypeId`	The ID of the HIT type of this HIT Type: String Default: None	No
`HITGroupId`	The ID of the HIT Group of this HIT Type: String Default: None	No
`HITLayoutId`	The ID of the HIT Layout of this HIT Type: String Default: None	No
`CreationTime`	The date and time the HIT was created Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as `2012-01-31T23:59:59Z`. Default: None	No
`Title`	The title of the HIT	No

Name	Description	Required
	Type: String Default: None	
`Description`	A general description of the HIT Type: String Default: None	No
`Keywords`	One or more words or phrases that describe the HIT, separated by commas. Search terms similar to the keywords of a HIT are more likely to have the HIT in the search results. Type: String Default: None	No
`HITStatus`	The status of the HIT and its assignments Type: String Valid Values: Assignable | Unassignable | Reviewable | Reviewing | Disposed Default: None	No
`Reward`	The amount of money the Requester will pay a Worker for successfully completing the HIT. Type: a `Price` data structure Default: None	No
`LifetimeInSeconds`	The amount of time, in seconds, after which the HIT is no longer available for users to accept. The HIT becomes unavailable even if the requested number of assignments, specified by `MaxAssignments`, has not been completed. Type: positive integer Default: None	No
`AssignmentDurationInSeconds`	The length of time, in seconds, that a Worker has to complete the HIT after accepting it. Type: positive integer Default: None	No

Name	Description	Required
`MaxAssignments`	The number of times the HIT can be accepted and completed before the HIT becomes unavailable. Type: positive integer Default: 1	No
`AutoApprovalDelayInSeconds`	The amount of time, in seconds, after the Worker submits an assignment for the HIT that the results are automatically approved by Amazon Mechanical Turk. This is the amount of time the Requester has to reject an assignment submitted by a Worker before the assignment is auto-approved and the Worker is paid. Type: positive integer Default: None	No
`Expiration`	The date and time the HIT expires Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as `2012-01-31T23:59:59Z`. Default: None	No
`QualificationRequirement`	The QualificationRequirement data structure describes a Qualification that a Worker must have before the Worker is allowed to accept a HIT. A requirement may optionally state that a Worker must have the Qualification in order to preview the HIT, or see the HIT in search results. A HIT can have between zero and ten Qualification requirements. Type: a `QualificationRequirement` data structure. Default: None	No
`Question`	The data the Worker completing the HIT uses produce the results. Type: either a QuestionForm (p. 97) or an ExternalQuestion (p. 92) data structure. Default: None	No

Name	Description	Required
`RequesterAnnotation`	An arbitrary data field the Requester who created the HIT can use. This field is visible only to the creator of the HIT. Type: String Default: None	No
`HITReviewStatus`	Indicates the review status of the HIT. Type: String Valid Values: NotReviewed | MarkedForReview | ReviewedAppropriate | ReviewedInappropriate Default: None	No
`NumberofAssignmentsPending`	The number of assignments for this HIT that are being previewed or have been accepted by Workers, but have not yet been submitted, returned, or abandoned. Type: non-negative integer Default: None Conditions: This element is returned only if the `HITAssignmentSummary` response group is specified.	Conditional
`NumberofAssignmentsAvailable`	The number of assignments for this HIT that are available for Workers to accept Type: non-negative integer Default: None Conditions: This element is returned only if the `HITAssignmentSummary` response group is specified.	Conditional
`NumberofAssignmentsCompleted`	The number of assignments for this HIT that have been approved or rejected. Type: non-negative integer Default: None Conditions: This element is returned only if the `HITAssignmentSummary` response group is specified.	Conditional

Example

The following example shows a HIT data structure returned by the `CreateHIT` operation. The `CreateHIT` operation returns an element named `HIT` that represents the HIT that was created by the call.

```
HIT:{
 HITId:"123RVWYBAZW00EXAMPLE",
 HITTypeId:"T100CN9P324W00EXAMPLE",
 HITTypeId:"2005-06-30T23:59:59",
 HITStatus:"Assignable",
 MaxAssignments:"5",
 AutoApprovalDelayInSeconds:"86400",
 LifetimeInSeconds:"86400",
 AssignmentDurationInSeconds:"300",
 Reward:{
  Amount:"25"
  CurrencyCode:"USD"
  FormattedPrice:"$0.25"
 },
 Title:"Location and Photograph Identification",
 Description:"Select the image that best represents...",
 Keywords:"location, photograph, image, identification, opinion",
 Question:{
  QuestionForm:[XML-encoded Question data]
 },
 QualificationRequirement:{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  Comparator:"GreaterThan",
  Value:"18"
 },
 HITReviewStatus:"NotReviewed"
}
```

HITLayoutParameter

Description

The `HITLayoutParameter` data structure defines parameter values used with a HITLayout. A HITLayout is a reusable Amazon Mechanical Turk project template used to provide Human Intelligence Task (HIT) question data for `CreateHIT`. For more information, see HITLayout (p. 86).

The `HITLayoutParameter` data structure is used in the results of the following operation:

- CreateHIT (p. 12)

Tip
The HITLayout (p. 86) is used to create an HTMLQuestion document. HITLayoutParameter values with reserved characters or invalid HTML markup may result in an invalid HTMLQuestion document.

Elements

The `HITLayout` data structure has a root element of `HITLayoutParameter`.

The `HITLayoutParameter` element contains the following elements:

Name	Description	Required
`Name`	The name of the parameter in the HITLayout. Type: String Default: None Constraints: Parameter names supplied in a HITLayoutParameter structure must be used in the referenced HITLayout.	Yes
`Value`	The value substituted for the parameter referenced in the HITLayout. Type: String Default: None	Yes

Qualification

Description

The Qualification data structure represents a Qualification assigned to a user, including the Qualification type and the value (score).

The Qualification data structure is used as a response element for the following operations:

- `GetQualificationScore`
- `ListQualificationRequests`

Elements

The Qualification structure can contain the elements described in the following table. When the structure is used in a request, elements described as **Required** must be included for the request to succeed.

Name	Description	Required
`QualificationTypeId`	The ID of the Qualification type for the Qualification. Can be up to 255 bytes in length. Type: String Default: None	Yes
`WorkerId`	The ID of the Worker who possesses the Qualification. Can be up to 255 bytes in length. Type: String Default: None	Yes
`GrantTime`	The date and time the Qualification was associated with the Worker. If the Worker's Qualification was revoked, and then re-associated based on a new Qualification request, `GrantTime` is the date and time of the last call to the AssociateQualificationWithWorker operation. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`** Default: None	Yes
`IntegerValue`	The value (score) of the Qualification, if the Qualification has an integer value. Type: Integer Default: None	No
`LocaleValue`	The value of the Qualification if the Qualification describes a geographical region or location.	No

Name	Description	Required
	Type: a Locale (p. 153) data structure. Default: None	
`Status`	The status of the Qualification Type: String Valid Values: Granted | Revoked Default: None	Yes

Example

The following example illustrates a Qualification with an integer value.

```
Qualification:{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  WorkerId:"AZ3456EXAMPLE",
  GrantTime:"2005-01-31T23:59:59Z",
  IntegerValue:95
}
```

QualificationRequest

Description

The QualificationRequest data structure represents a request a Worker has made for a Qualification.

The QualificationRequest data structure is used as a response element for the following operations:

- `ListQualificationRequests`

Elements

The QualificationRequest structure can contain the elements described in the following table:

Name	Description	Required
`QualificationRequestId`	The ID of the Qualification request, a unique identifier generated when the request was submitted. Can be up to 255 bytes in length. Type: String Default: None	No
`QualificationTypeId`	The ID of the Qualification type the Worker is requesting, as returned by the `CreateQualificationType` operation. Can be up to 255 bytes in length. Type: String Default: None	No
`SubjectId`	The ID of the Worker requesting the Qualification. This ID corresponds to the `WorkerId` returned with assignment results when the Worker performs a HIT. Can be up to 255 bytes in length. Type: String Default: None	No
`Test`	The contents of the Qualification test that was presented to the Worker, if the type has a test and the Worker has submitted answers. This value is identical to the QuestionForm associated with the Qualification type at the time the Worker requests the Qualification. Type: a QuestionForm (p. 97) data structure Default: None	No
`Answer`	The Worker's answers for the Qualification type's test contained in a `QuestionFormAnswers` document, if the type has a test and the Worker has submitted answers. If the Worker does not provide any answers, `Answer` may be empty.	No

Name	Description	Required
	Type: a QuestionFormAnswers (p. 113) data structure Default: None	
SubmitTime	The date and time the Qualification request had a status of **Submitted**. This is either the time the Worker submitted answers for a Qualification test, or the time the Worker requested the Qualification if the Qualification type does not have a test. Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **2005-01-31T23:59:59Z** Default: None	No

Example

The following example shows a QualificationRequest data structure returned by the ListQualificationRequests operation. This operation returns the requests for Qualifications of a Qualification type to the owner of the type.

```
QualificationRequest:{
  QualificationRequestId:"789RVWYBAZW00EXAMPLE951RVWYBAZW00EXAMPLE",
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  SubjectId:"AZ3456EXAMPLE",
  Test:{
   QuestionForm:[XML-encoded Question data]
 },
 Answer:{
  QuestionFormAnswers:[XML-encoded Answer data]
 },
  SubmitTime:"2005-12-01T23:59:59Z"
}
```

QualificationRequirement

Topics

Description

The QualificationRequirement data structure describes a Qualification that a Worker must have before the Worker is allowed to accept a HIT. A requirement may optionally state that a Worker must have the Qualification in order to preview the HIT, or see the HIT in search results.

The QualificationRequirement data structure is used as a parameter for the following operations:

- `CreateHIT`
- `CreateHITType`

The QualificationRequirement data structure is used in the HIT data structure (p. 128).

Using Custom, System-Assigned, and Master Qualification Types

A Qualification requirement can be based on a Qualification you assign to Workers. You can create a custom Qualification type using the `CreateQualificationType` operation. Then you can grant requests for the Qualification automatically using a Qualification test and answer key submitted with the Qualification type, or you can grant the request manually with the `AcceptQualificationRequest` operation. The `CreateQualificationType` returns a `QualificationTypeId`, which you can use with the `QualificationRequirement` data structure to identify the type of Qualification the Worker is required to have to accept a HIT. Either the Qualification test or your call to `AcceptQualificationRequest` determines a Qualification value, which is compared to the requirement in the HIT to determine the Worker's eligibility.

Amazon Mechanical Turk supplies several Qualification types for use by all Requesters. Mechanical Turk system-assigned Qualification types work the same way as Qualifications that you create, except that data from the Mechanical Turk marketplace determines the Worker's values. Every Worker has a value for each system Qualification, and these values are updated as the Worker uses the system. Additionally, Amazon Mechanical Turk also provides Master Qualification types that give you access to an elite group of Workers who have demonstrated superior performance while completing thousands of

HITs across the Mechanical Turk marketplace. You can use the Master and system-assigned Qualification types by using the corresponding Qualification type ID in the `QualificationTypeId` element of the `QualificationRequirement` data structure. For a list of the Master and system-assigned IDs, see Qualification Type IDs (p. 142). For more information about using a Master Qualification type, see Master Qualification (p. 143).

Elements

The QualificationRequirement data structure can contain the following elements.

Name	Description	Value
`QualificationTypeId`	The ID of the Qualification type for the requirement. Can be up to 255 bytes in length.	A valid QualificationType ID from a custom Qualification type you created or an ID from the list of Master and system-assigned Qualification Type IDs (p. 142).
`Comparator`	The kind of comparison to make against a Qualification's value. You can compare a Qualification's value: • To an `IntegerValue` to see if it is `LessThan`, `LessThanOrEqualTo`, `GreaterThan`, `GreaterThanOrEqualTo`, `EqualTo`, or `NotEqualTo` the `IntegerValue`. • To a `LocaleValue` to see if it is `EqualTo`, or `NotEqualTo` the `LocaleValue`. • To see if the value is `In` or `NotIn` a set of `IntegerValue` or `LocaleValue` values. A Qualification requirement can also test if a Qualification `Exists` or `DoesNotExist` in the user's profile, regardless of its value.	`LessThan` | `LessThanOrEqualTo` | `GreaterThan` | `GreaterThanOrEqualTo` | `EqualTo` | `NotEqualTo` | `Exists` | `DoesNotExist` | `In` | `NotIn`
`IntegerValues`	Array of integer values to compare against the Qualification's value. `IntegerValue` must not be present if `Comparator` is `Exists` or `DoesNotExist`. `IntegerValue` can only be used if the Qualification type has an integer value; it cannot be used with the `Worker_Locale` QualificationType ID, see Qualification Type IDs (p. 142). When performing a set comparison by using the `In` or the `NotIn` comparator, you can use up to 15 elements in this list. For an example of a set comparison, see Example—Using the QualificationRequirement Data Structure for Comparing Multiple Values (p. 145).	An integer.

Name	Description	Value
`LocaleValue`	The locale value to compare against the Qualification's value. The local value must be a valid ISO 3166 country code or supports ISO 3166-2 subdivisions. `LocaleValue` can only be used with a `Worker_Locale` QualificationType ID, see Qualification Type IDs (p. 142). `LocaleValue` can only be used with the `EqualTo`, `NotEqualTo`, `In`, and `NotIn` comparators. You must only use a single `LocaleValue` element when using the `EqualTo` or `NotEqualTo` comparators. When performing a set comparison by using the `In` or the `NotIn` comparator, you can use up to 30 `LocaleValue` elements in a QualificationRequirement data structure. For an example of a set comparison, see Example —Using the QualificationRequirement Data Structure for Comparing Multiple Values (p. 145).	A Locale (p. 153) data structure.

Name	Description	Value
`ActionsGuarded`	Setting this attribute prevents Workers whose Qualifications do not meet this requirement from taking the specified action. Valid arguments are: • `Accept` (worker cannot accept the HIT) • `PreviewAndAccept` (worker cannot accept or preview the HIT) • `DiscoverPreviewAndAccept` (worker cannot accept, preview, or see the HIT in their search results) It's possible for you to create a HIT with multiple QualificationRequirements (which can have different values for the ActionGuarded attribute). In this case, the Worker is only permitted to perform an action when they have met all QualificationRequirements guarding the action. The actions in the order of least restrictive to most restrictive is Discover, Preview and Accept. For example, if a Worker meets all QualificationRequirements that are set to DiscoverPreviewAndAccept, but do not meet all requirements that are set with PreviewAndAccept, then the Worker will be able to Discover, i.e. see the HIT in their search result, but will not be able to Preview or Accept the HIT. The default is `Accept`.	Accept | PreviewAndAccept | DiscoverPreviewAndAccept
`RequiredToPreview`	DEPRECATED as of 03/29/2018 - Use the `ActionsGuarded` field instead. If **`true`**, the question data for the HIT will not be shown when a Worker whose Qualifications do not meet this requirement tries to preview the HIT. That is, a Worker's Qualifications must meet all of the requirements for which `RequiredToPreview` is **`true`** in order to preview the HIT. If a Worker meets all of the requirements where `RequiredToPreview` is **`true`** (or if there are no such requirements), but does not meet all of the requirements for the HIT, the Worker will be allowed to preview the HIT's question data, but will not be allowed to accept and complete the HIT. The default is **`false`**.	A Boolean value, **`true`** or **`false`**

Qualification Type IDs

The following table lists the Master and system-assigned Qualification Type IDs that can be used in the `QualificationTypeId` element.

Name	QualificationTypeId	Description
Masters	Sandbox: 2ARFPLSP75KLA8M8DH1HTEQVJT3SY6 Production: 2F1QJWKUDD8XADTFD2Q0G6UTO95ALH	Masters are Workers who have demonstrated superior performance while completing thousands of HITs across the Mechanical Turk marketplace. Masters maintain this high level of performance to keep this distinction. Set the `comparator` parameter to "Exists" to require that Workers have this Qualification. Note that for this Qualification type ID, the `QualificationTypeId` value for the Mechanical Turk Sandbox environment is different than the value for the production environment.
Worker_ NumberHITsApproved	00000000000000000040	Specifies the total number of HITs submitted by a Worker that have been approved. The value is an integer greater than or equal to 0.
Worker_Locale	00000000000000000071	The location of the Worker, as specified in the Worker's mailing address. For more information about the locale Qualification, see the The Locale Qualification (p. 144) section.
Worker_Adult	00000000000000000060	Requires workers to acknowledge that they are over 18 and that they agree to work on potentially offensive content. The value type is boolean, 1 (required), 0 (not required, the default).
Worker_ PercentAssignmentsApproved	000000000000000000L0	The percentage of assignments the Worker has submitted that were subsequently approved by the Requester, over all assignments the Worker has submitted. The value is an integer between 0 and 100. Note that a Worker's approval rate is statistically meaningless for small numbers of assignments, since a single rejection can reduce the approval rate by many percentage points. So to ensure that a new Worker's approval rate is unaffected by these statistically meaningless changes, if a Worker has submitted

Name	QualificationTypeId	Description
		less than 100 assignments, the Worker's approval rate in the system is 100%. To prevent Workers who have less than 100 approved assignments from working on your HIT, set the Worker_NumberHITsApproved qualification type ID to a value greater than 100.

Master Qualification

You can require that Workers must have a Master Qualification to complete your HITs.

To create a Qualification requirement for Masters, specify:

- A `QualificationTypeId` of 2F1QJWKUDD8XADTFD2Q0G6UTO95ALH
- A `Comparator` of `Exists`

> **Note**
> The Master Qualification Type ID values used for the `QualificationTypeId` parameter in the preceding procedures are for the production environment. The ID values to use in the Mechanical Turk Sandbox environment are listed in the Qualification Type IDs (p. 142) table.

Adding Adult Content

Adult content can be offensive to some people. For that reason, if your HIT is adult-oriented, we ask you to use the following procedure.

Adding Adult HITs

1. In the HIT title, include the words "adult content."
2. Specify the worker's qualifications in one of the following ways:

Using the API: • Set the `CreateHit` parameter, `QualificationRequirement`, to the qualification type, 00000000000000000060. • Set `comparator` parameter to "EqualTo." • Set the `IntegerValue` parameter to 1 (required).

3. Define the HIT to be private or previewed.

   This setting prevents anyone who does not qualify from seeing the HIT. To make the HIT private, use one of the following methods:

Using the API, set the `ActionsGuarded` parameter to `PreviewAndAccept`.
Using the command line tools, in the HIT properties file, set the private parameter, `qualification.private`, to TRUE.

The Locale Qualification

You can create a Qualification requirement based on the Worker's location. The Worker's location is specified by the Worker to Amazon Mechanical Turk when the Worker creates his account.

To create a Qualification requirement based on the Worker's location, specify:

- A `QualificationTypeId` of 00000000000000000071
- A `Comparator` of `EqualTo` or `NotEqualTo`
- A `LocaleValue` that corresponds to the desired locale

To create a Qualification requirement based on the Worker being in or not in one of several locations, specify:

- A `QualificationTypeId` of 00000000000000000071
- A `Comparator` of `In` or `NotIn`
- Multiple `LocaleValue` values that correspond to the desired locales.

For more information on the format of a `LocaleValue` element, see Locale data structure (p. 153).

Example

The following example shows a QualificationRequirement specifying Workers located in the state of Pennsylvania. Workers located in the state of New York (US-NY) or in the country of India (IN) do not qualify for the HITs with this QualificationRequirement.

```
QualificationRequirement:{
   QualificationTypeId:"00000000000000000071",
   Comparator:"EqualTo",
   LocaleValues:[{
    Country:"US",
    Subdivision:"PA"
 }]
}
```

The following example shows a QualificationRequirement specifying Workers not located in the state of California. Workers located in the state of New York or in the country of India (IN) will be qualified to do HITs with this QualificationRequirement.

```
QualificationRequirement:{
   QualificationTypeId:"00000000000000000071",
   Comparator:"NotEqualTo",
   LocaleValues:[{
    Country:"US",
    Subdivision:"CA"
 }]
}
```

Example—Using the QualificationRequirement Data Structure

The following example of a QualificationRequirement data structure could be passed in to a call to `CreateHIT`. `CreateHIT` accepts parameters that describe the HIT being created, including one or more Qualification requirements:

```
QualificationRequirement:{
 QualificationTypeId:"789RVWYBAZW00EXAMPLE",
 Comparator:"GreaterThan",
 IntegerValues:[18]
}
```

Example—Using the QualificationRequirement Data Structure for Comparing Multiple Values

The following example of a QualificationRequirement data structure could be passed in to a call to `CreateHIT`. `CreateHIT` accepts parameters that describe the HIT being created, including one or more Qualification requirements.

Example Request

The following example shows a QualificationRequirement data structure used in a SOAP request that uses multiple `IntegerValue` values when performing a set comparison by using the `In` comparator.

```
QualificationRequirement:{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  Comparator:"In",
  IntegerValues:[10, 20, 30]
}
```

Example—Using the QualificationRequirement Data Structure to Hide HIT from Unqualified Workers

The following example of a QualificationRequirement data structure could be passed in to a call to `CreateHIT`. This will hide the HIT from unqualified Workers (these Workers will not be able to accept, preview, or see the HIT in search results). `CreateHIT` accepts parameters that describe the HIT being created, including one or more Qualification requirements:

```
QualificationRequirement:{
 QualificationTypeId:"789RVWYBAZW00EXAMPLE",
 Comparator:"Exists",
 ActionsGuarded:"DiscoverPreviewAndAccept"
}
```

QualificationType

Description

The QualificationType data structure represents a Qualification type, a description of a property of a Worker that must match the requirements of a HIT for the Worker to be able to accept the HIT. The type also describes how a Worker can obtain a Qualification of that type, such as through a Qualification test.

The QualificationType data structure is used as a response element for the following operations:

- `CreateQualificationType`
- `GetQualificationType`
- `ListQualificationTypes`
- `UpdateQualificationType`

Elements

The QualificationType structure can contain the elements described in the following table:

Name	Description	Required
`QualificationTypeId`	A unique identifier for the Qualification type. A Qualification type is given a Qualification type ID when you call the `CreateQualificationType` operation operation, and it retains that ID forever. Can be up to 255 bytes in length. Type: String Default: None	No
`CreationTime`	The date and time the Qualification type was created Type: a dateTime structure in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as **`2005-01-31T23:59:59Z`**. Default: None	No
`Name`	The name of the Qualification type. The type name is used to identify the type, and to find the type using a Qualification type search. Type: String Default: None	No
`Description`	A long description for the Qualification type. Type: String Default: None	No
`Keywords`	One or more words or phrases that describe theQualification type, separated by commas. The	No

Name	Description	Required
	Keywords make the type easier to find using a search. Type: String Default: None	
`QualificationTypeStatus`	The status of the Qualification type. A Qualification type's status determines if users can apply to receive a Qualification of this type, and if HITs can be created with requirements based on this type. Type: String Valid Values: Active | Inactive Default: None	No
`RetryDelayInSeconds`	The amount of time, in seconds, Workers must wait after taking the Qualification test before they can take it again. Workers can take a Qualification test multiple times if they were not granted the Qualification from a previous attempt, or if the test offers a gradient score and they want a better score. Type: positive integer Default: None. If not specified, retries are disabled and Workers can request a Qualification only once.	No
`Test`	The questions for a Qualification test associated with this Qualification type that a user can take to obtain a Qualification of this type. Type: a QuestionForm (p. 97) data structure. **Note** A Qualification test cannot use an ExternalQuestionQuestionForm (p. 92) like a HIT can. Default: None Constraints: must be specified if `AnswerKey` is present. A Qualification type cannot have both a specified `Test` parameter and an `AutoGranted` value of **`true`**.	No
`TestDurationInSeconds`	The amount of time, in seconds, given to a Worker to complete the Qualification test, beginning from the time the Worker requests the Qualification. Type: positive integer Default: None	No

Name	Description	Required
AnswerKey	The answers to the Qualification test specified in the Test parameter. Type: an AnswerKey (p. 119) data structure. Default: None. If not provided with a test, the Qualification author must process the Qualification request manually.	No
AutoGranted	Specifies that requests for the Qualification type are granted immediately, without prompting the Worker with a Qualification test. Type: Boolean Valid Values: true | false Default: None Constraints: A Qualification type cannot have both a specified Test parameter and an AutoGranted value of **true**.	No
AutoGrantedValue	The Qualification value to use for automatically granted Qualifications, if AutoGranted is true. Type: Integer Default: 1	No
IsRequestable	Specifies whether the Qualification type is one that a user can request through the Amazon Mechanical Turk web site, such as by taking a Qualification test. This value is **false** for Qualifications assigned automatically by the system. Type: Boolean Valid Values: true | false Default: None	No

Example

The following example shows a QualificationType data structure returned by a call to the ListQualificationTypes operation. The GetQualificationType operation returns a QualificationType element.

```
QualificationType:{
  QualificationTypeId:"789RVWYBAZW00EXAMPLE",
  CreationTime:"2005-01-31T23:59:59Z",
  Name:"EnglishWritingAbility",
  Description:"The ability to write and edit text...",
  Keywords:"English, text, write, edit, language",
  QualificationTypeStatus:"Active",
  RetryDelayInSeconds:86400,
```

```
  IsRequestable:true
}
```

HIT Review Policy

Topics

Description

HIT Review Policy data structures represent HIT review policies, which you specify when you create a Human Intelligence Task (HIT). For more information about Review Policies, see Review Policies (p. 155).

The following two types of HIT Review Policy data structures are used when calling the CreateHIT (p. 12) operation.

- `AssignmentReviewPolicy` data structures set the review results and actions at the Assignment level. For more information, see Assignment Review Policies (p. 156).
- `HITReviewPolicy` data structures set the review results and actions at the HIT-level. For more information, see HIT Review Policies (p. 158).

The HIT Review Policy data structure is used in the following operation:

- `CreateHIT`

HIT Review Policy Elements

The HIT Review Policy data structures can contain the following elements.

Name	Description	Required
`PolicyName`	Name of a Review Policy. For policy names and descriptions, see Assignment Review Policies (p. 156) and HIT Review Policies (p. 158). Type: String Constraints: SimplePlurality/2011-09-01 or ScoreMyKnownAnswers/2011-09-01	Yes
`Parameter`	Name of the parameter from the Review policy. Type: Parameter data structure, for a description see the next section Parameter Elements.	

Parameter Elements

The Parameter data structure contains the elements described in the following table.

Name	Description	Required
`Key`	Name of the parameter from the list of Review Polices. For a list of valid parameter names, see Assignment Review Policies (p. 156) and HIT Review Policies (p. 158). Type: String Constraints: none Default: none	
`Value`	Value of the parameter. Type: Varies.	
`MapEntry`	This data structure is the data type for the `AnswerKey` parameter of the ScoreMyKnownAnswers/2011-09-01 Review Policy, see Assignment Review Policies (p. 156). Type: MapEntry data structure, for a description, see the next section MapEntry Elements.	

MapEntry Elements

The MapEntry element contains the elements described in the following table.

Name	Description	Required
`Key`	The QuestionID from the HIT that is used to identify which question requires Mechanical Turk to score as part of the ScoreMyKnownAnswers/2011-09-01 Review Policy. Can be up to 255 bytes in length. Type: String none	
`Value`	The answer to the question specified in the MapEntry `Key` element. Note: You can provide multiple `Value` elements, but the Worker must match all values in order for the answer to be scored correctly. Type: String	

Examples

The following example request shows the structure of a `CreateHIT` request using HIT Review Policy data structures. The example request is followed by an example response.

Sample Request

The following is an example `CreateHIT` request.

```
{ HITTypeId:"T100CN9P324W00EXAMPLE",
    Question:URL-encoded question data,
    LifetimeInSeconds:604800,
    AssignmentReviewPolicy:{
        PolicyName:"ScoreMyKnownAnswers/2011-09-01",
        Parameters:[
   { Key:"AnswerKey",
      MapEntries:[
     {Key: "QuestionId1", Values:["B"]},
     {Key: "QuestionId2", Value:["A"]},
     {Key: "QuestionId3", Value:["C"]},
     {Key: "QuestionId4", Value:["A"]}
     ]
   },
   { Key: "ApproveIfKnownAnswerScoreIsAtLeast",
     Values:["79"]
   }
   { Key: "ExtendIfKnownAnswerScoreIsLessThan",
     Values:["79"]
   }
   { Key: "ExtendMaximumAssignments",
     Values:["3"]
   }
  ]
 },
    HITReviewPolicy:{
        PolicyName:"SimplePlurality/2011-09-01",
        Parameters:[
   {
    Key:"QuestionIDs",
    // Add upto 15 questions
    Values:["questionid1", "questionid2", "questionid3", "questionid4", "questionid5"]
   },
   {
    Key:"QuestionAgreementThreshold",
    Values:["100"]
   }
  ]
 }
}
```

Locale

Description

The Locale data structure represents a geographical region or location.

The Locale data structure is used as part of the QualificationRequirement (p. 138) data structure when you specify a requirement based on the locale Qualification, and as part of the Qualification (p. 134) data structure that describes the value of a locale Qualification.

When used in a QualificationRequirement, the Locale data structure only needs to contain as much of the locale as the Worker needs to match to meet the requirement. For example, a requirement that the Worker be living anywhere in the United States would have only the `Country` field.

Note
Currently, a Locale data structure only supports the `Country` field and `Subdivision` field. Please note that subdivisions or states are only available for the United States of America.

Elements

The Locale structure can contain the elements described in the following table. When the structure is used in a request, elements described as **Required** must be included for the request to succeed.

Name	Description	Required
`Country`	The country of the locale. Type: A valid ISO 3166 country code. For example, the code **`US`** refers to the United States of America. Default: none	Yes
`Subdivision`	The state or subdivision of the locale. Type: Type: A valid ISO 3166-2 subdivision code. For example, the code CA refers to the state of California Default: none	No

Example

The following code sample indicates a locale in the United States.

```
LocaleValue:{
  Country:"US"
}
```

Example

The following code sample indicates a locale in the state of California in the United States of America.

```
LocaleValue:{
  Country:"US",
  Subdivision:"CA"
}
```

Review Policies

Using Amazon Mechanical Turk Review Policies you can evaluate Worker submissions against a defined set of criteria. You specify the Review Policy(s) that you want to use when you call the CreateHIT (p. 12) operation.

There are two types of Review Policies, Assignment-level and HIT-level:

- An Assignment-level Review Policy is applied as soon as a Worker submits an assignment. For more information, see Assignment Review Policies (p. 156).
- A HIT-level Review Policy is applied when a HIT becomes reviewable. For more information, see HIT Review Policies (p. 158).

You can select from a set of pre-defined Review Policies. One Review Policy leverages *known answers* or *gold standards* within a Human Intelligence Task (HIT) and has Mechanical Turk calculate a Worker's performance on these known answers. You can specify an action for Mechanical Turk to take automatically based on Worker performance against the known answer.

Mechanical Turk has Review Policies that calculate consensus/agreement among multiple Workers performing the same HITs. For instance, you can specify a Review Policy that measures agreement on work items within the HIT and authorizes Mechanical Turk to keep asking additional Workers to work on the HIT, until a certain level of agreement is achieved. Once the required level of agreement is achieved, the results are returned to you for immediate use.

Review Policies that track Worker performance on your known answers and agreement with other Workers give you information you can use to manage your Workers. For more information about using Review Policies, see Review Policy Use Cases (p. 162).

How Review Policies Work

You specify the Review Policy(s) that you want Mechanical Turk to apply when you call the CreateHIT (p. 12) operation. You must specify Review Policies when you create a HIT. You cannot apply a Review Policy to an existing HIT.

As assignments are submitted, Mechanical Turk applies the Review Policy(s) that you specify. You call the ListReviewPolicyResultsForHIT operation to gather the results from the application of the Review Policy.

There are two types of Review Policies, Assignment-level Review Policies that are applied as soon as a Worker submits an assignment and HIT-level Review Policies that are applied when a HIT becomes reviewable. For more information, see Assignment Review Policies (p. 156) and HIT Review Policies (p. 158).

You can specify one Assignment-level Review Policy and one HIT-level Review Policy when you call `CreateHIT` using the HIT Review Policy (p. 150) data structure. The Assignment-level Review Policy ScoreMyKnownAnswer/2011-09-01 and the HIT-level Review Policy SimplePlurality/2011-09-01 can be used in the same call to `CreateHIT`.

Once an Assignment-level Review Policy is applied, the Assignment's status is changed to *Submitted* and optionally an event notification can be sent. Assignments with *Submitted* status are returned by the ListAssignmentsForHIT operation and the results of applying the Review Policy are available by using the ListReviewPolicyResultsForHIToperation.

You can use different Review Policies on distinct HITs in a HIT type. For example, you may wish to apply the ScoreMyKnownAnswers/2011-09-01 policy to a small number of HITs that have known answers in them, but apply the SimplePlurality/2011-09-01 policy to all HITs in a group. Workers do not have access on the Worker User Interface to information about whether a Review Policy has been applied to a HIT.

Assignment Review Policies

Assignment-level Review Policies are applied as soon as a Worker submits an assignment.

ScoreMyKnownAnswers/2011-09-01

ScoreMyKnownAnswers/2011-09-01 is an Assignment-level Review Policy.

Description

You can use the ScoreMyKnownAnswers/2011-09-01 Review Policy for QuestionForm (QAP) HITs and for ExternalQuestion (iframe) HITs. You provide an answer key when you call the CreateHIT (p. 12) operation. The answer key is a collection of QuestionIds, where each QuestionId has a set of zero or more values that represent the correct response for that QuestionId. For more information about QuestionForm and ExternalQuestion HITs, see QuestionForm (p. 97) and ExternalQuestion (p. 92).

You can specify if one question in your HIT has a known answer or if many questions in your HIT have known answers. When a Worker submits an assignment Mechanical Turk examines the Worker's answers and compares them against the set of known answers that you provide when you create the HIT. Mechanical Turk then calculates a score, for example, 4 out of 10 known answers were correct.

Based on how the Worker's level of agreement with the known answers compares with various configurable thresholds, Mechanical Turk can automatically take actions you requested to approve the assignment, automatically reject the assignment, or automatically extend the HIT to publish an assignment for another Worker.

A Worker's performance on known answers within a specific assignment are returned from calling the ListReviewPolicyResultsForHIT operation.

Mechanical Turk evaluates answers and considers the following answers as not matching:

- The Worker left an empty value set in the answer key.
- The answer key has an empty value set but the Worker supplied an answer.
- The Worker provides an answer that is the wrong case or has incorrect punctuation that doesn't match the answer exactly. You can either use structured HTML form elements to restrict the values a Worker can submit, or use JavaScript to validate and normalize the submitted values.
- The answer key says a question's answer is A and B but the Worker's value is A.
- The answer key says a question's answer is A and the Worker selected both A and B.

When comparing answers for a match, Mechanical Turk removes any whitespace from before and after the Worker's answer, and from before and after the answer you provide.

Parameters

The following parameters are specified in the AssignmentReviewPolicy element when calling the `CreateHIT` operation. You must also specify the PolicyName *ScoreYourKnownAnswers/2011-09-01*

as part of the AssignmentReviewPolicy element. For an example of how to structure the AssignmentReviewPolicy element, see the HIT Review Policy (p. 150) data structure.

Name	Description	Required
AnswerKey	Question IDs and the answers to the questions. Type: MapEntry, see the HIT Review Policy (p. 150) data structure. Default: None	Yes
ApproveIfKnownAnswerScoreIsAtLeast	Approve the assignment if the KnownAnswerScore is equal to or greater than this value. If not specified, assignments are left in the submitted state and are not approved or rejected. Type: Integer Constraints: Minimum value 0 (always approve), maximum 101 (never approve)	No
ApproveReason	A description provided to the Worker about the reason the assignment was approved. If not specified, the reason is left blank. Type: String	No
RejectIfKnownAnswerScoreIsLessThan	Reject the assignment if the KnownAnswerScore is equal to or less than this value. If not specified, assignments are left in the submitted state and are not approved or rejected. Type: Integer Constraints: Minimum value 0 (never reject), maximum 101 (always reject).	No
RejectReason	A description provided to the Worker about the reason the assignment was rejected. If not specified, the reason is left blank. Type: String	No
ExtendIfKnownAnswerScoreIsLessThan	Extend the HIT by one assignment to allow one more Worker to complete it if the known answer score is less than this value. Ordinarily this is done to replace an assignment that is being rejected or that is not usable because the Worker didn't answer the known answer correctly. If omitted the HIT is not extended. Type: String	No

Name	Description	Required
	Constraint: Minimum value 0 (never extend), maximum 101 (always extend).	
`ExtendMaximumAssignments`	The maximum number of assignments the HIT can be extended. Note that if you use the CreateAdditionalAssignmentsForHIT operation and specify a maximum assignment count greater than this value, ScoreMyKnownAnswers will not extend the HIT. **Note**: If a HIT is created with fewer than 10 assignments, it will not extend to have 10 or more assignments. Type: Integer Constraint: Minimum value 2, maximum 25. Default: 5	No
`ExtendMinimumTimeInSeconds`	The additional time in seconds to let other Workers complete the extended assignment. Type: Integer Constraints: Minimum of 60 (one minute), Maximum of 31536000 (one year). Default: 0	No

HIT Review Policies

A HIT-level Review Policy is applied when a Human Intelligence Task (HIT) becomes reviewable.

SimplePlurality/2011-09-01

SimplePlurality/2011-09-01 is a HIT-level Review Policy.

Description

The SimplePlurality/2011-09-01 policy allows you to automatically compare answers received from multiple Workers and detect if there is a majority or consensus answer. The results can optionally trigger additional actions, such as approving the assignments that matched the majority answer. The results of this comparison are available as a part of the ListReviewPolicyResultsForHIT operation.

Mechanical Turk evaluates answers and considers the following answers as not matching:

- The Worker provides an answer that is the wrong case or incorrect punctuation that doesn't match the answer exactly to another Worker. You can either use structured HTML form elements to restrict the values a Worker can submit, or use JavaScript to validate and normalize the submitted values.
- One Worker's answer is A and B, but another Worker's value is A.
- One Worker's answer is A, but another Worker selected both A and B.

When comparing answers for a match, Mechanical Turk removes any whitespace from before and after the Worker's answer.

Note
Answers that are longer than 256 characters are not used in the computation of HIT review policies.

Parameters

The following parameters are specified in the HITReviewPolicy element when calling the CreateHIT (p. 12) operation. You must also specify the PolicyName *SimplePlurality/2011-09-01* as part of the HitReviewPolicy element. For an example, see HIT Review Policy (p. 150) data structure.

Name	Description	Required
`QuestionIds`	A comma-separated list of questionIds used to determine agreement. Type: String Constraints: none	Yes
`QuestionAgreementThreshold`	If the Question Agreement Score is greater than this value, the questionId is considered to have an *agreed answer*. Type: Integer Constraints: none	Yes
`DisregardAssignmentIfRejected`	Excludes rejected assignments from agreement calculation. Type: Boolean Constraints: T or F	Yes
`DisregardAssignmentIfKnownAnswerScoreIsLessThan`	Excludes answers from agreement calculation if the KnownAnswerScore is present and less than the provided value. Type: Integer Constraints: none	No
`ExtendIfHITAgreementScoreIsLessThan`	If the HIT Agreement Score is less than this value, extend the HIT to another Worker to complete. If omitted, extending on failure is disabled. Type: Integer Constraints: 1-100	No
`ExtendMaximumAssignments`	If the ExtendIfHITAgreementScoreIsLessThan is provided, this sets the total maximum number of assignments for the HIT.	Conditional

Name	Description	Required
	If you use ExtendHIT operation and specify the maximum assignment count greater than this value, ScoreMyKnownAnswers will not extend the HIT. **Note**: If a HIT is created with fewer than 10 assignments, it will not extend to have 10 or more assignments. Type: Integer Constraints: none Conditions: Required if ExtendIfHITAgreementScoreIsLessThan is provided.	
`ExtendMinimumTimeInSeconds`	If the ExtendIfHITAgreementScoreIsLessThan is provided, this sets the additional time that the HIT will be extended by. Type: Integer Constraints: Minimum of 60 (one minute), Maximum of 31536000 (365 days) Conditions: Required if ExtendIfHITAgreementScoreIsLessThan is provided.	Conditional
`ApproveIfWorkerAgreementScoreIsAtLeast`	If the Worker Agreement Score is not less than this value, approve the Worker's assignment. If omitted, assignment will not be approved or rejected. Type: Integer Constraints: none	No
`RejectIfWorkerAgreementScoreIsLessThan`	If the Worker Agreement Score is less than this value, reject the Worker's assignment. If omitted, assignment will not be approved or rejected. Type: Integer Constraints: none	No

Name	Description	Required
`RejectReason`	If the RejectIfWorkerAgreementIsScoreLessThan value is provided, this value sets the reason for any automated rejections. Type: String Constraints: none	Optional

Scores

The following scores are calculated data from the SimplePlurality/2011-09-01 policy. Based on the value of these scores, Mechanical Turk can take various actions that you specify in the `CreateHIT` operation. It is important to understand how these scores are calculated so you can specify the appropriate actions to take, including approving or rejecting assignments, or extending HITs. The following chart describes how the scores are calculated.

Score	Description
Question Agreement Score	Percentage of Workers who provided the agreed-upon answer for a HIT. Note: Answer values are not normalized for case, whitespace, or punctuation before comparison. Answers can contain multiple values (such as in a set of check boxes); two answers agree with each other if they have the same values present and absent. We don't recommend using free format answers because values are not normalized.
HIT Agreement Score	Percentage of questions within the HIT with an agreed-upon answer. The number of questions within the HIT with an agreed-upon answer, divided by the number of questions evaluated.
Worker Agreement Score	The percentage of questions to which a Worker's answer agreed with other Workers' answers in the same HIT. If a question does not have an agreed upon answer the question is disregarded in this calculation.

The example chart below describes how the Answer Agreement Score and Worker Agreement Score is calculated for a HIT with 4 questions and answers from 3 Workers.

QuestionId	Worker1's answers	Worker2's answers	Worker3's answers	Has Agreed-upon value?	Agreed-upon value	Question Agreement Score
A	coat	sweater	coat	Yes	coat	66%
B	blue	blue	green	Yes	blue	66%
C	large	large	large	Yes	large	100%
D	Furry	fur	furr	No	n/a	n/a
Worker Agreement Score	100%	66%	66%			

The Question Agreement Score for questions A and B are 66% because two Workers agreed on the same answer. The HIT Agreement Score for this HIT is 75%. The HIT had four questions, and three of them had an agreed-upon answer for a percentage of 75%. The Worker Agreement Score for Worker 1 is 100% because this Worker agreed with the other Workers for each answer, except Question D where there was no conclusive answer.

Review Policy Use Cases

The following use cases show you how to apply ScoreYourKnownAnswers and SimplePlurality policies when you call the CreateHIT (p. 12) operation.

Photo Moderation Use Case – Single Worker with Known Answers

In this scenario, you want Workers to moderate photos and screen the photos for inappropriate content. You place 20 photos in a single HIT and 5 of the 20 photos are your known answers. You are using Master Workers and have created the HIT with only one initial assignment. You want to use the answers based on the Worker getting at least 4 of the 5 known answers (80% Answer Agreement Score) correct. If the first Worker does not meet the Answer Agreement score of 80%, then you want to extend the HIT to another Worker. But, in this scenario, you only want to extend the HIT to a maximum of three Workers.

Elements and Parameters

The following is a list of elements and parameters you need to specify in the CreateHIT (p. 12) operation to execute the above scenario and allow Mechanical Turk to automatically calculate the known answer score. Note that this `CreateHIT` example assumes you have already created a HIT Type.

Element	Parameter	Value
`AssignmentReviewPolicy`	PolicyName	ScoreMyKnownAnswers/2011/09/01
`AssignmentReviewPolicy`	AnswerKey	List of questionIDs and answers.
`AssignmentReviewPolicy`	ExtendIfKnownAnswerScoreIsLessThan	80
`AssignmentReviewPolicy`	ExtendMaximumAssignments	3

Examples

The following example shows how to use the above elements and parameters with the `CreateHIT` operation.

Sample CreateHIT Request

The following example shows a `CreateHIT` request.

```
<CreateHITRequest>
    <HITTypeId>T100CN9P324W00EXAMPLE</HITTypeId>
    <Question>[CDATA block or XML Entity encoded]</Question>
```

```
    <LifetimeInSeconds>604800</LifetimeInSeconds>
    <AssignmentReviewPolicy>
        <PolicyName>ScoreMyKnownAnswers/2011-09-01</PolicyName>
        <Parameter>
            <Key>AnswerKey</Key>
            <MapEntry>
                <Key>QuestionId3</Key>   <!—correct answer is “B” -->
                <Value>B</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId7</Key>  <!—correct answer is “A” -->
                <Value>A</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId15</Key>   <!—correct answer is “F” -->
                <Value>F</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId17</Key>   <!—correct answer is “C” -->
                <Value>C</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId18</Key>   <!—correct answer is “A” -->
                <Value>A</Value>
            </MapEntry>
        </Parameter>
        <Parameter>
           <Key>ExtendIfKnownAnswerScoreIsLessThan</Key>
           <Value>80</Value>
        </Parameter>
        <Parameter>
           <Key>ExtendMaximumAssignments</Key>
           <Value>3</Value>
        </Parameter>
    </AssignmentReviewPolicy>
</CreateHITRequest>
```

Photo Moderation Use Case – Multiple Workers with Agreement

In this scenario, you want Workers to moderate photos and screen the photos for inappropriate content. You place 20 photos in a single HIT and 5 of the 20 photos are your known answers. You want to approve the assignment if the Worker completes at least 4 of the 5 known answers correct (at least 80% Answer Agreement Score).

You want 3 Workers to complete each HIT and you want to calculate the HIT Agreement Score for the 15 photos you don’t know the answer to. Also, you want to disregard the Worker's answer in the Agreement Score if they don't get 4 of 5 of the known answers correct.

Elements and Parameters

The following is a list of elements and parameters you need to specify in the CreateHIT (p. 12) operation to execute the above scenario and allow Mechanical Turk to automatically approve the assignments. Note that this CreateHIT example assumes you have already created a HIT Type.

Element	Parameter	Value
AssignmentReviewPolicy	PolicyName	ScoreMyKnownAnswers/2011/09/01

Element	Parameter	Value
AssignmentReviewPolicy	Answer	List of questionIDs and answers.
AssignmentReviewPolicy	ApproveIfKnownAnswerScoreIsAtLeast	80
AssignmentReviewPolicy	ExtendIfKnownAnswerScoreIsLessThan	80
AssignmentReviewPolicy	ExtendMaximumAssignments	3
HITReviewPolicy	PolicyName	SimplePlurality/2011-09-01
HITReviewPolicy	QuestionIDs	Your list of 15 question IDs.
HITReviewPolicy	QuestionAgreementThreshold	100
HITReviewPolicy	DisregardAssignmentIfKnownAnswerScoreIsLessThan	80

Examples

The following example shows how to use the above elements and parameters with the CreateHIT operation.

Sample CreateHIT Request

The following example shows a CreateHIT request.

```
<CreateHITRequest>
    <HITTypeId>T100CN9P324W00EXAMPLE</HITTypeId>
    <Question>[CDATA block or XML Entity encoded]</Question>
    <LifetimeInSeconds>604800</LifetimeInSeconds>
    <AssignmentReviewPolicy>
        <PolicyName>ScoreMyKnownAnswers/2011-09-01</PolicyName>
        <Parameter>
            <Key>AnswerKey</Key>
            <MapEntry>
                <Key>QuestionId3</Key>   <!—correct answer is “B” -->
                <Value>B</Value>
            </MapEntry
            <MapEntry>
                <Key>QuestionId4</Key>  <!—correct answer is “A” -->
                <Value>A</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId13</Key>   <!—correct answer is “F” -->
                <Value>F</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId14</Key>   <!—correct answer is “C” -->
                <Value>C</Value>
            </MapEntry>
            <MapEntry>
                <Key>QuestionId19</Key>   <!—correct answer is “A” -->
                <Value>A</Value>
            </MapEntry>
         </Parameter>
        <Parameter>
           <Key>ApproveIfKnownAnswerScoreIsAtLeast</Key>
           <Value>80</Value>
        </Parameter>
```

```
        <Parameter>
           <Key>ExtendIfKnownAnswerScoreIsLessThan</Key>
           <Value>80</Value>
        </Parameter>
        <Parameter>
           <Key>ExtendMaximumAssignments</Key>
           <Value>3</Value>
        </Parameter>
    </AssignmentReviewPolicy>
    <HITReviewPolicy>
        <PolicyName>SimplePlurality/2011-09-01</PolicyName>
        <Parameter>
           <Key>QuestionIDs</Key>
           <Value>questionid1</Value>
           <Value>questionid2</Value>
           <Value>questionid5</Value>
           <Value>questionid6</Value>
           <Value>questionid7</Value>
                 ..... <! Add your additional 10 questionIDs for a total of 15 questions.
 Different from your known answer questionIDs.>
        </Parameter>
        <Parameter>
           <Key>QuestionAgreementThreshold</Key>
           <Value>100</Value>
        </Parameter>
        <Parameter>
           <Key>DisregardAssignmentIfKnownAnswerScoreIsLessThan</Key>
           <Value>80</Value>
        </Parameter>
    </HITReviewPolicy>
</CreateHITRequest>
```

Categorization and Tagging Use Case – Multiple Workers

In this scenario, you want Workers to categorize a product and provide multiple tags for the product in a HIT. You also want the Workers to be able to comment on your HIT and give you feedback.

You want to calculate the Answer Agreement Score for only the categorization question. If two Workers do not agree on the product categorization question, you want to extend the HIT to a third Worker. Also, you want to extend the assignment by an hour so the third Worker has time to work on the assignment.

Elements and Parameters

The following is a list of elements and parameters you need to specify in the CreateHIT (p. 12) operation to execute the above scenario and allow Mechanical Turk to automatically calculate agreement and approve or reject the assignments. Note that this CreateHIT example assumes you have already created a HIT Type.

Element	Parameter	Value
`HITReviewPolicy`	PolicyName	SimplePlurality/2011-09-01
`HITReviewPolicy`	QuestionIDs	questionID1
`HITReviewPolicy`	QuestionAgreementThreshold	100
`HITReviewPolicy`	ExtendMinimumTimeInSeconds	3600

Element	Parameter	Value
HITReviewPolicy	ExtendMaximumAssignments	3

Examples

The following example shows how to use the above elements and parameters with the CreateHIT operation.

Sample CreateHIT Request

The following example shows a CreateHIT request.

```
<CreateHITRequest>
    <HITTypeId>T100CN9P324W00EXAMPLE</HITTypeId>
    <Question>[CDATA block or XML Entity encoded]</Question>
    <LifetimeInSeconds>604800</LifetimeInSeconds>
    <HITReviewPolicy>
        <PolicyName>SimplePlurality/2011-09-01</PolicyName>
        <Parameter>
           <Key>QuestionIDs</Key>
           <Value>questionID1</Value>
        </Parameter>
        <Parameter>
           <Key>QuestionAgreementThreshold</Key>
           <Value>100</Value>
        </Parameter>
        <Parameter>
           <Key>ExtendMaximumAssignments</Key>
           <Value>3</Value>
        </Parameter>
        <Parameter>
           <Key>ExtendMinimumTimeInSeconds</Key>
           <Value>3600</Value>
        </Parameter>
    </HITReviewPolicy>
</CreateHITRequest>
```

Managing Notifications

Topics

This section describes how to set up and handle Amazon Mechanical Turk event notification messages. A notification message describes one or more events that happened in regards to a HIT type. For more information, see Elements of a Notification Message (p. 167).

You can configure Amazon Mechanical Turk to notify you whenever certain events occur during the life cycle of a HIT. Mechanical Turk can send you a notification message when a Worker accepts, abandons, returns, or submits an assignment, when a HIT becomes "reviewable", or when a HIT expires, for any HIT of a given HIT type.

Notifications are specified as part of a HIT type. To set up notifications for a HIT type, you call the UpdateNotificationSettings (p. 75) operation with a HIT type ID and a notification specification. For more information about HIT types, see Understanding HIT Types.

A notification specification is defined by a Notification (p. 174) data structure, which describes a HIT event notification for the HIT type. The notification specification is passed as the `Notification` parameter when calling UpdateNotificationSettings (p. 75).

Amazon Mechanical Turk can send a notification to an Amazon Simple Queue Service (Amazon SQS) queue or to an Amazon Simple Notification Service (Amazon SNS) topic.

For more information about setting up and handling notifications, see Creating and Managing Notifications.

For more information on configuring notifications using SQS, see Notification Handling Using Amazon SQS (p. 168).

For more information on configuring notifications using SNS, see Notification Handling Using Amazon SNS (p. 171).

You can test your application's ability to receive notifications using SendTestEventNotification (p. 71).

Elements of a Notification Message

Notification messages contain one or more `Event` data structures that describe recent activity for HITs of a HIT type.

The Notification API Version

Similar to how a REST request that is sent to the Amazon Mechanical Turk Requester service must include a `Version` parameter to indicate which version of the service API the client is expecting to use, a notification message must also include a `Version` parameter. This version string is identical to the version that is included in the notification specification for the HIT type.

Tip
Your application may need to accommodate receiving notification messages of different versions at the same time if you want to upgrade your notification specifications to a new version without missing messages. You can avoid having to accommodate multiple API versions by first disabling the notification specifications that use the old version, upgrading your application to use the new version, then updating the notification specifications to use the new version and re-enable notifications.

When a new version of the notification API is made available, all existing notification specifications will continue to use the API versions they were using previously. You must update your notification specifications to use a new version of the API.

Events

A notification message describes one or more events that happened in regards to a HIT type. Each event includes:

- the event type (`EventType`), a value corresponding to the `EventType` value in the notification specification data structure (p. 174)
- the time of the event (`EventTime`), as a dateTime in the Coordinated Universal Time time zone, such as **`2005-01-31T23:59:59Z`**
- the HIT type ID for the event (`HITTypeId`)
- the HIT ID for the event (`HITId`)
- the assignment ID for the event, if applicable (`AssignmentId`)

Multiple events may be batched into a single notification message.

Notification Handling Using Amazon SQS

Your application can use the Amazon Simple Queue Service (Amazon SQS) to handle Mechanical Turk notifications. By using Amazon SQS, your notifications are guaranteed to be delivered at least once. For more information about guaranteed delivery of notifications, see Guaranteed Delivery (p. 170). For more information about, see Amazon SQS.

Creating an SQS Queue

You must create an Amazon SQS queue before using the SQS transport type in notification-related calls. Mechanical Turk does not create an Amazon SQS queue for you. An SQS queue can be created through the Amazon SQS API or by using the AWS Console. For more information, see the Amazon SQS documentation.

Configuring an SQS Queue

Your Amazon SQS queue permissions must be configured to allow a Mechanical Turk system account to call the `sqs:SendMessage` action on your queue. Whether you use the management console UI or the API to configure permissions, consider the following:

- You must add a permission that enables the Mechanical Turk service principal **`mturk-requester.amazonaws.com`** to call SendMessage on your queue.
- Your SendMessage permission must add an action of `aws:SecureTransport` set to **`true`**.
- Limit the permissions you apply to this queue to those that will actually be used.

- You should consider disallowing all other access to your queue from other accounts.

  This makes it easy for you to be sure that available messages were sent by Mechanical Turk.

  If you enable SendMessage for other accounts to this queue, or if you plan to send messages to this queue from your AWS account, you should check the sending identity for every message that you receive from the queue. You can do this by requesting the `SenderId` attribute in your call to ReceiveMessage. This value will be **`AIDAIXO4EZE6RHVSXIN4E`**. Amazon SQS provides this value as a strong guarantee of the authenticated identity of the sender, so if it matches, you can be sure the message came from Mechanical Turk.

  For more information, see the Amazon SQS Developer Guide and Amazon SQS API Reference.

Amazon SQS Policy Document Example

The following example policy document only creates the SendMessage permission for the Mechanical Turk account. You can add additional restrictions. For more information about policy documents, see the Amazon SQS Developer Guide.

```
{
  "Version": "2012-10-17",
  "Id": "arn:aws:sqs:region:aws-account-id:queue-name/MTurkOnlyPolicy",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "mturk-requester.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:region:aws-account-id:queue-name",
      "Condition": {
        "Bool": {
          "aws:SecureTransport":"true"
        }
      }
    }
  ]
}
```

Configuring Permissions Using the AWS Console

To configure permissions in the AWS Console:

1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Select your queue, and then select **Permissions**.
3. Click **Edit Policy Document**.
4. Enter a policy document similar to the example.

Configuring Permissions Using the Amazon SQS API

Call the Amazon SQS SetQueueAttributes action with the `Attribute.Name` parameter set to **`Policy`**. You can call `SetQueueAttributes` with a policy document similar to the example policy document. Do not use the Amazon SQS `AddPermission` action for configuring permissions on this queue. If you

programmatically create a queue and apply a policy document to it, you must ensure the `Resource` value in the policy document is updated with the correct queue name.

Testing Your Queue

To test your permissions, call the Mechanical Turk SendTestEventNotification (p. 71) operation with a `Transport` of **`SQS`** and your queue URL as the `Destination`.

Guaranteed Delivery

Using Amazon SQS provides a guaranteed at-least-once delivery of each message. Mechanical Turk ensures that it calls SendMessage at least once for each message. SQS then provides guarantees regarding message persistence and message delivery.

Rarely, the same message may show up twice in the queue. This is an attribute of Amazon SQS's nature as a distributed system.

If you take action on your queue that prevents Mechanical Turk from publishing to it, we cannot guarantee delivery of the messages that would have been sent to your queue. For instance, such actions may include:

- Modifying the permissions on your queue in a way that prevents our account from calling SendMessage successfully.
- Deleting or disabling your queue.

SQS Message Ordering

You should expect that messages may arrive out of order. For information about message ordering behavior, see the SQS documentation.

Multiple SQS Queues

You may use a different queue for each HITType that you configure with notifications.

Mechanical Turk does not provide the ability to route events within a HITType to different queues. For example, you might prefer to have AssignmentSubmitted events for a HITType delivered to a different queue than HITReviewable events for that same HITType. Mechanical Turk will publish both events to the same queue. You can split the events into different queues by running an SQS client that pulls the messages and republishes them to different queues depending on the event type.

SQS Message Payload

The body of each SQS message is a JSON-encoded structure that provides support for multiple events in each message.

The JSON-encoded structure contains the following:

- EventDocVersion: This is the requested version that is passed in the call to UpdateNotificationSettings (p. 75), such as **`2014-08-15`**. For a requested version, Mechanical Turk will not change the structure or definition of the output payload structure in a way that is not backward-compatible.
- EventDocId: A unique identifier for the Mechanical Turk event. In rare cases, you may receive two different SQS messages for the same event, which can be detected by tracking the EventDocId values you have already seen.

- CustomerId: Your Customer Id.
- Events: A list of Event structures, described next.

The Event structure contains the following:

- EventType: A value corresponding to the EventType value in the notification specification data structure.
- EventTimestamp: A dateTime in the Coordinated Universal Time time zone, such as **`2005-01-31T23:59:59Z`**.
- HITTypeId: The HIT type ID for the event.
- HITId: The HIT ID for the event.
- AssignmentId: The assignment ID for the event, if applicable.

Double Delivery

Amazon SQS already provides a `MessageId` value that enables double-delivery detection in the typical SQS case. However, when receiving messages from Mechanical Turk, we recommend that you use the EventDocId value for double-delivery detection. This will cover an additional scenario in which you may see the same EventDocId in two messages with distinct MessageIds.

Most messages are safe to process twice, since they represent independent one-way state changes. Consider whether detection of repeated messages is important for your application. You may be able to simply process the message and ignore it if it appears to have been applied already.

Notification Handling Using Amazon SNS

Your application can use the Amazon Simple Notification Service (Amazon SNS) to handle Mechanical Turk notifications. For more information about Amazon SNS, see Amazon SNS.

Creating an SNS Topic

You must create an Amazon SNS topic before using the SNS transport type in notification-related calls. Mechanical Turk does not create an Amazon SNS topic for you. An SNS topic can be created through the Amazon SNS API or by using the AWS Console. For more information, see the Amazon SNS documentation.

Configuring an SNS Topic

Your Amazon SNS topic permissions must be configured to allow a Mechanical Turk system account to publish to your topic. Whether you use the management console UI or the API to configure permissions, consider the following:

- You must add a permission that enables the Mechanical Turk service principal **`mturk-requester.amazonaws.com`** to Publish to your topic.
- You should ensure that only notifications from your Mechanical Turk account can be published to your topic. This can be done using a `StringEquals` IAM Policy Condition for the IAM Policy Condition Key `aws:SourceAccount` in your SNS Topic Policy doc. Set the `aws:SourceAccount` value equal to the AWS Account Id that is linked to your Mechanical Turk account.

  You can determine the AWS Account Id that is linked to your Mechanical Turk account by visiting the Mechanical Turk Developer page.

For more information on the use of IAM Policy Conditions, see the IAM Policy Condition Element documentation.

- Your Publish permission must add an action of `aws:SecureTransport` set to **`true`**.
- Limit the permissions you apply to this topic to those that will actually be used.
- You should consider disallowing all other access to your topic from other accounts.

  This makes it easy for you to be sure that all messages were sent by Mechanical Turk.

  For more information, see the Amazon SNS Developer Guide and Amazon SNS API Reference.

Amazon SNS Policy Document Example

The following example policy document only creates the Publish permission for the Mechanical Turk account. You can add additional restrictions. For more information about policy documents, see the Amazon SNS Developer Guide.

```
{
  "Version": "2012-10-17",
  "Id": "arn:aws:sns:region:aws-account-id:topic-name/MTurkOnlyPolicy",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "mturk-requester.amazonaws.com"
      },
      "Action": "SNS:Publish",
      "Resource": "arn:aws:sns:region:aws-account-id:topic-name",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "linked-aws-account-id"
        },
        "Bool": {
          "aws:SecureTransport":"true"
        }
      }
    }
  ]
}
```

Configuring Permissions Using the AWS Console

To configure permissions in the AWS Console:

1. Sign in to the AWS Management Console and open the Amazon SNS console at https://console.aws.amazon.com/sns/.
2. Select your topic, and then select **Actions**.
3. Click **Edit Topic Policy**.
4. Enter a policy document similar to the example.

Configuring Permissions Using the Amazon SNS API

Call the Amazon SNS SetTopicAttributes action with the `AttributeName` parameter set to **`Policy`**. You can call `SetTopicAttributes` with a policy document similar to the example policy document.

Do not use the Amazon SNS `AddPermission` action for configuring permissions on this topic. If you programmatically create a topic and apply a policy document to it, you must ensure the `Resource` value in the policy document is updated with the correct topic name.

Testing Your Topic

To test your permissions, call the Mechanical Turk SendTestEventNotification (p. 71) operation with a `Transport` of **`SNS`** and your topic ARN as the `Destination`.

SNS Message Payload

The body of each SNS message is a JSON-encoded structure that provides support for multiple events in each message.

The JSON-encoded structure contains the following:

- EventDocVersion: This is the requested version that is passed in the call to UpdateNotificationSettings (p. 75), such as **`2014-08-15`**. For a requested version, Mechanical Turk will not change the structure or definition of the output payload structure in a way that is not backward-compatible.
- EventDocId: A unique identifier for the Mechanical Turk event. In rare cases, you may receive two different SNS messages for the same event, which can be detected by tracking the EventDocId values you have already seen.
- CustomerId: Your Customer Id.
- Events: A list of Event structures, described next.

The Event structure contains the following:

- EventType: A value corresponding to the EventType value in the notification specification data structure.
- EventTimestamp: A dateTime in the Coordinated Universal Time time zone, such as **`2005-01-31T23:59:59Z`**.
- HITTypeId: The HIT type ID for the event.
- HITId: The HIT ID for the event.
- AssignmentId: The assignment ID for the event, if applicable.

Double Delivery

When receiving messages from Mechanical Turk, we recommend that you use the `EventDocId` value for double-delivery detection.

Most messages are safe to process twice, since they represent independent one-way state changes. Consider whether detection of repeated messages is important for your application. You may be able to simply process the message and ignore it if it appears to have been applied already.

Notification

Description

The Notification data structure describes a HIT event notification for a HIT type.

Elements

The Notification structure can contain the elements described in the following table. When the structure is used in a request, elements described as **Required** must be included for the request to succeed.

Name	Description	Required
`Destination`	The destination for notification messages. Type: String • For Amazon Simple Queue Service (Amazon SQS) notifications (if `Transport` is **`SQS`**), this is the URL for your Amazon SQS queue. For more information, see Notification Handling Using Amazon SQS (p. 168). • For Amazon Simple Notification Service (Amazon SNS) notifications (if `Transport` is **`SNS`**), this is the ARN for your Amazon SNS topic. For more information, see Notification Handling Using Amazon SNS (p. 171). Default: None	Yes
`Transport`	The method Amazon Mechanical Turk uses to send the notification. Type: String Valid Values: SQS | SNS Default: None	Yes
`Version`	The version of the Notification data structure schema to use. Type: String Valid Values: 2014-08-15 Default: None	Yes
`EventTypes`	The array of one or more events that should cause notifications to be sent. The Ping event is only valid for the SendTestEventNotification (p. 71) operation. Type: Array of Strings Valid Values: AssignmentAccepted | AssignmentAbandoned | AssignmentReturned | AssignmentSubmitted | AssignmentRejected |	Yes

Name	Description	Required
	AssignmentApproved | HITCreated | HITExtended | HITDisposed | HITReviewable | HITExpired | Ping Default: None	

Example

In the following example, the notification specification specifies that an event notification message will be published to an SNS topic when a Worker accepts a HIT.

```
{
  Destination:"arn:aws:sns:us-east-1:7429088EXAMPLE:my_mturk_topic",
  Transport: "SNS",
  Version:"2014-08-15",
  EventTypes:["AssignmentAccepted"]
}
```