Interface Specific for Download API of Android Extension

**（javascript）**

1. DownloadManager

**The DownloadManager provides a single entry point to interact with the downloading capabilities, including starting new downloads, controlling ongoing downloads and retrieving downloads of different services.**

Properties：**none**

Methods：**( each method returns a promise )**

1. **createDownload( properties ) : creates a new** Download **object**

|  |
| --- |
| Promise<Download> createDownload ( properties ); |
| Var properties = {  “source” : url,  “target”: path};  Var download;  DownloadManager.createDownload( properties ). then( function (data) {  // create the download successfully  download = data; }, function (error){  //fail to create the download  }); |

**The parameter** properties **can be a** JSON **type, containing the** source **and the** target **of the download.**

1. **getAll() : gets an array of** Download **objects that are currently in the system**

|  |
| --- |
| Promise<Array<Download>> getAll( ); |
| Var downloads = new Array() ;  DownloadManager.getAll( ). then( function (data) {  //get all downloads successfully  data.results.forEach( function (downloadId){  var i = 0;  downloads[i] = downloadId;  i++;  }); }, function (error){  //failed to get all downloads  }); |

1. **getDownloadById( downloadID ) : gets a** Download **object of the same** downloadID **as the parameter**

|  |
| --- |
| Promise< Download> getDownloadById( downloadID ); |
| Var downloadID = 10;  Var download;  DownloadManager.getDownloadById( downloadID ). then( function (data) {  // get the download by ID successfully  download = data.result; }, function (error){  //fail to get the download by ID  }); |

1. **getDownloadsByState( state ) : gets an array of** Download **objects of the same** state **as the parameter**

**The parameter** state **can be “**running**” “**paused**” “**canceled**” “**failed**” “**successful**”.**

|  |
| --- |
| Promise< Download> getDownloadsByState( state ); |
| Var state = “paused”;  Var downloads = new Array();  DownloadManager.getDownloadsByState( state ). then( function (data) {  //get the downloads by status successfully  data.results.forEach( function (downloadId){  var i = 0;  downloads[i] = downloadId;  i++;  }); }, function (error){  //failed to get the downloads by status  }); |

1. **removeDownloads( downloadIDs ) : removes some** Download **objects whose** downloadIDs **are within the parameter.**

|  |
| --- |
| Promise removeDownloads( downloadIDs ); |
| var IDs = new Array();  IDs[0] =19;  IDs[1] = 12;  DownloadManager.removeDownloads ( IDs). then( function (data) {  // remove the downloads successfully }, function (error){  //failed to remove the downloads  }); |

**The parameter** downloadIDs **can be an** array **type**

1. **removeAll() : removes all** Download **objects in the system**

|  |
| --- |
| Promise<Array<Download>> removeAll( ); |
| DownloadManager.removeAll().then( function (data) {  // remove all downloads successfully }, function (error){  //failed to remove all downloads  }); |

1. Download

**A** Download **object represents a single download, with associated state and actions.**

Properties：

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | number | The identifier of the download |
| source | String | Source of download |
| target | String | Target file of download |
| startTime | Date | The start time of the download |
| state | String | Indicates the current state of the download job. It can be**“**running**” “**paused**” “**canceled**” “**failed**” “**successful**”.** |
| error | JSON  //enum | The causes of failure |
| totalBytes | Number | The total bytes of the download file |
| MIMEType | String | The MIME type of download |

Methods：**(all methods return promise)**

1. **start ( progressCallback ) ：starts the download for the first time, or restarts a download that failed or has been canceled. This method can call back the ongoing progress in a regular interval.**

**The parameter** callback **is a** function **that the developers use for getting the progress of the download.**

|  |
| --- |
| Promise start( progressCallback ); |
| //case 1  Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  download.start( function ( progress) {  //call back for getting the progress of the download  }).then( function (data) {  //start the download successfully }, function (error){  //failed to start the download  }); |
| //case 2  Var properties = {  “source” : url,  “target”: path};  Var download = DownloadManager.createDownload( properties );  download.start( function ( progress) {  //call back for getting the progress of the download  }).then( function (data) {  //start the download successfully }, function (error){  //failed to start the download  }); |

1. **pause() : pauses the download when the download is progressing**

|  |
| --- |
| Promise pause( ); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  download.pause( ).then( function (data) {  // pause the download successfully }, function (error){  //failed to pause the download  }); |

1. **resume() : resumes the download when it has been paused**

|  |
| --- |
| Promise resume( ); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  //if(download.state == “paused” )  download.resume().then( function (data) {  //resume the download successfully }, function (error){  //failed to resume the download  }); |

1. **cancel() : cancels the download**

|  |
| --- |
| Promise cancel( ); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  //if(download.state != “canceled” )  download.cancel().then( function (data) {  // cancel the download successfully }, function (error){  //failed to cancel the download  }); |

1. **showContainingDirectory( ) : shows the folder containing the target file**

|  |
| --- |
| Promise showContainingDirectory(); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  if(download.state == “succeeded” )  download.showContainingDirectory().then( function (data) {  // open the directory successfully }, function (error){  //failed to open the directory  }); |

1. **open() : opens the file of the download**

|  |
| --- |
| Promise open( ); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  if(download.state == “succeeded” )  download. open().then( function (data) {  // open the download file successfully }, function (error){  //failed to open the download file  }); |

1. **removePartialData() : removes any partial data kept as part of a canceled or failed download**

|  |
| --- |
| Promise removePartialData( ); |
| Var downloadID = 10;  Var download = DownloadManager.getDownload( downloadID );  if(download.state == “canceled” )  download.removePartialData().then( function (data) {  // remove the partial data of the download successfully }, function (error){  //failed to remove the partial data of the download  }); |