



3D Juump Infinite web API usage

[WU_EN] version 3.1

Introduction	4
Connection	6
1.1 - Directory session	6
1.2 - Data session.....	7
Metadata	9
1.1 - Id card.....	9
1.2 - Attribute dictionary	10
1.3 - Project document.....	10
Context	11
1.1 - Configuration context	11
1.2 - Visibility context.....	12
Filtering.....	13
1.1 - Filter solver	13
1.2 - Filter All parts.....	13
1.3 - Filter AABB (Axis Aligned Bounding Box).....	14
1.4 - Filter part instance list.....	14
1.5 - Filter Attribute.....	14
1.6 - Filter Range	14
1.7 - Filter Has Field	14
1.8 - Filter Boolean.....	14
1.9 - Filter Literal	14
1.10 - Filter set.....	14
1.11 - Filter Compound	15
Search	16
Materials.....	18
Navigation	19
Converter.....	20
1.1 - Part converter	20

- 1.2 - Part instance converter.....21
- 1.3 - Geometric instance converter22
- Rendering primitives.....24**
 - 1.1 - Point24
 - 1.2 - Line25
 - 1.3 - Box.....25
 - 1.4 - Primitive materials26

Introduction

This document will describe the basic element provided by the 3D Juump Infinite web API and the method to use it. The first step is to create a web API engine object which is the main object providing all services. To do this, creates a metadata manager, a directory session and optionnaly a infinite cache object. The metadata manager will manage all metadata providing by the server such as part informations, configurations, documents. It also provide a search service, converter services and manage the filtering of the DMU. The directory session object manage the user identification to the server. And the infinite cache provide a cache services to increase loading performance.

```
mInfiniteCache = InfiniteCacheFactory.CreateInfiniteCache(lCacheHDSize
, lCacheBuildSize);
mLoaderManager = LoaderManagerFactory.CreateLoaderManager(mInfiniteCache);
mMetadataManager = MetadataManagerFactory.CreateMetadataManager(mLoaderManager);
mDirectorySession = DirectorySessionFactory.CreateDirectorySession( `directory host`, `directory path`, `directory port`, `id token name`);
mInfiniteEngine = InfiniteFactory.CreateInfiniteEngine(mMetadataManager, mDirectorySession, mInfiniteCache);
```

Then if you want to visualize the 3D scene, sets a HTML container where the scene will be rendered with `setView`.

```
let lView3D: HTMLElement = <HTMLElement>document.getElementById("view3D");
mInfiniteEngine.setView(lView3D);
```

Objects in the API implements an interface `EventDispatcherInterface` to register event listeners on the event target. When a event is fired the listeners callback associated is notified.

```
let onReady = (event: InfiniteEvent, pCallbackData: Object | undefined) => {
    // TODO
}

eventTarget.addEventListener("ready", onReady);
```

Connection

To access to 3D Juump Infinite you need to be identified. The identification system is deferred to project owner company using OpenId protocol. The connection system is composed of two objects *directorySession* and *DataSession*. The first allows to identify the user to the associated directory server. The second allows you to access to the data of the digital mockup.

1.1 - Directory session

First step is to create a directory session object using *DirectorySessionFactory.CreateDirectorySession* with given the directory host, api access path, port and a name to store the identification token in the local storage. Identification uses OpenId connect protocol, so you need to provide a redirect url which be called after identification succes. Then create the authentication url with *getAuthenticationUrl* and open it.

```
let lDirectorySession = DirectorySessionFactory.CreateDirectorySession
(host, path, port, key);

let lRedirectUrl = window.location.origin + lPath.join("/") + "/authen
ticate.html";
let lURL = lDirectorySession.getAuthenticationUrl(sRedirectUrl);
if (lURL !== "" && lURL.indexOf("authenticate") > 0) {
    window.open(lURL);
}
```

After authentication succes the redirect url is open, then you have to parse the content of `window.Location.hash`, to do so use `DecodeAuthenticationHash` function.

```
if (!window.Location.hash) {
    return;
}

if (!DirectorySessionFactory.DecodeAuthenticationHash(window.Location
.hash)) {
    console.log("Error: unable to decode hash content of authenticatio
n");
}
else {
    window.close();
}
```

After this step, user is authenticated to the directory and the event `LoginSuccess` is fired (the event `LoginFailed` or `LoginAborted` may be fired if authentication failed or aborted). On login success callback, the attachment contains a list of build, tags user, teams to which he belongs and information about the user (name, initials and a picture). The api provide a function `ConvertBuildMapToProjectsInformation` which gather build in to projects list (ie all build of the same project is gather together)

```
//Register callback before start authentication
lInfiniteEngine.addEventListener("loginSuccess", onLoginSuccess);
lInfiniteEngine.addEventListener("loginFailed", onLoginFailed);
lInfiniteEngine.addEventListener("loginAborted", onLoginAbort);
...

//Define login succes callback
let onLoginSuccess = (pEvent: InfiniteEvent) => {
    //Get attachements
    let lAttachments: ConnectionData = pEvent.attachments;

    // Convert to project list
    lProjectList = ConvertBuildMapToProjectsInformation(lAttachments.b
uilds);
    ...
}
```

1.2 - Data session

After authentication, to get acces to the data of a build, creates a data session with `createDataSession` on directory session object. This function is asynchronous and the result is given with the succes callback. On data session creation, gets the data session and open a connection to the selected build with `openDb` on the infinite engine object. Event `onOpenDBSuccess` is fired by infinite engine interface if the database is open.

```
lInfiniteEngine.addEventListener("openDBSuccess", onOpenDBSuccess);
lDirectorySession.createDataSession(lBuildId, onDataSessionSuccess, on
DataSessionError, onDataSessionAbort);
...
const onDataSessionSuccess = (event: InfiniteEvent) => {
    lDataSession = <DataSessionInterface>event.emitter;
```

```
    lInfiniteEngine.openDb(lDataSession);  
};  
...  
let onOpenDBSuccess = (pEvent: InfiniteEvent, pCallbackData: Object |  
undefined) => {  
    //Do something  
};
```


Metadata

1.1 - Id card

To retrieve the informations about a part instance (and not a geometric instance id) such as the metadata of the part and link, the list of part instance ancestors, attached document, annotations, ... creates an id card getter object with `createIdCardGetter` function from the metadata manager interface. Then set a list of part instance ids with the `retrieveIdCard` from the id card getter interface. When the result is done, a `ready` event is fired. Retrieves informations with `getPartInstanceInfos` function.

```
mMetadataManagerInterface = mEngineInterface.getMetadataManagerInterface();
mIdCardGetter = mMetadataManagerInterface.createIdCardGetter();
mIdCardGetter.addListener("ready", onIdCardGetterReady);
mIdCardGetter.retrieveIdCard( part instance ids)

...

let onIdCardGetterReady = (pEvent: InfiniteEvent) => {
    let lIdCardGetter: IdCardGetterInterface = <IdCardGetterInterface>
    pEvent.emitter;
    if (lIdCardGetter.getLastError().length != 0) {
        return;
    }
}
```

```

    let lPartInstanceInfos = lIdCardGetter.getPartInstanceInfos();
    if( (lPartInstanceInfos == undefined) ||
        (lPartInstanceInfos.length == 0) ){
        let lError = lIdCardGetter.getLastErrorMessage();
        return lError;
    }

    //Show id card
}

```

1.2 - Attribute dictionary

To retrieve the list of attributes defined in the digital mock-up, get the attribute dictionary from the metadata manager interface. Then you can access to the dictionary with [getDictionary](#) function. All attributes is defined by:

- The name of the attribute as set by the connector. This is the attribute as seen by couchdb.
- If this attribute is part of a subdocument (nested) mNestedPath is path of nested array in parent. Undefined if unspecified
- Elastic Search Format (used by the mapping). Undefined if unspecified
- The type of the attribute
- Gets list of values existing in the database. If undefined the number of values exceeded the directory enumeration limit.
- Gets the minimum value of the attribute (only for Attribute of type number and date)
- Gets the maximum value of the attribute (only for Attribute of type number and date)

1.3 - Project document

Gets access to project document define by the project maintainer with [getProjectDocument](#) function from the metadata manager interface. This document is accesible with the internal id which is:

- [com.3djump:scripts](#) customisation scripts created by the project maintaine
- [com.3djump:defaultsettings](#) settings used as default to configure the client
- [com.3djump:indexinfo](#) list of attributes (use Attribute dictionary instead)

Context

1.1 - Configuration context

Interface used to set Configuration filtering. The configuration context does fetch the list of the current geometric instance ids. To instantiate a Configuration context use the metadata manager.

```
mMetadataManager = mEngineInterface.getMetadataManagerInterface();  
mConfContext = mMetadataManager.createConfContext();  
mConfContext.addListener("ready", onConfContextReady);
```

Then to set the active configuration

```
mConfContext.setActiveConfs([list of configuration id])
```

A configuration is created by the project maintainer and made available to all users. The list of available Configuration is given by the metadata manager function [getConfigurationList](#). A configuration is define

- The id of the configuration. This is used by the ConfContextInterface to switch configuration.
- The GUI name of the configuration (as created by the project maintener)
- The description of the configuration. A short text explaining the configuration

- The effectivity of the configuration. The list of rules that were used to create the configuration.

After each changes of active confs, call the method `update` on the metadata manager. An event `ready` is fired when the computation is finished. On registered callback you can retrieve the result.

```
let onConfCtxReady = (pEvent) => {  
  let lConfContext = pEvent.emitter;  
  let lGeometries = lConfContext.getGeometricInstanceIds();  
  ...  
}
```

1.2 - Visibility context

Interface used to limit the next filtering request to some specific parts. The visibility context is basically a list of filter solvers. The resulting parts will be the union of all the solvers inside this context. The visibility context may be used to restrict further filtering requests to a subset of the DMU. This object cannot be used explicitly and is not updated on demand. It neither holds any data. The only way to know the content of a visibility context is to iterate over its solvers and sets the visibility properties accordingly. To instanciate a visibility context use the metadata manager.

```
mMetadataManager = mEngineInterface.getMetadataManagerInterface();  
mVisibilityContext = mMetadataManager.createVisibilityContext();
```

Sets the optional configuration context used to compute filtering data with `setConfContext` function. The parts inside the visibilty will also be filtered by the active configuration. Inserts or removes filter solver with `insertFilterSolver` and `removeFilterSolver` function to manage the visibility context.

Filtering

To filter digital mockup, uses the creation functions of filters. This filters allow to filter in and out data from some criteria. The usage is to create a visibility context and optionnaly a configuration context. To set the configuration context of the visibility context. You will then add a list of filter solver to the visibility context. To create a filter solver use the [createFilterSolver](#) function. After each modification of the filter solver or filter content call the function [update](#) on metadata manager interface to update and recompute the filtering result.

1.1 - Filter solver

A filter solver is a combination of multiple filters with a specific operator (union/intersection/exclusion), a FilterOperator. The order of filters in the solver are therefore relevant and changing the order of the filters may (or not) change the final result. A filter solver is represented as a set of {geometric ids} and {part ids}. The FilterOperator is included in the filter (and not the filter solver), and represents the set operator to use when combined with the FORMER filter in the list of the filters of the filter solver. As such, it is COMPULSORY to set the filter operator of the first filter to FO_UNION, as using any other operator will result in a filter with no instance.

1.2 - Filter All parts

The easiest filter. This filter selects all the parts of the DMU.

1.3 - Filter AABB (Axis Aligned Bounding Box)

A filter to select part intersecting an axis aligned box. The inclusion algorithm may be strict, i.e. all triangles of a part must be strictly contained inside the box for the part to be elected, or loose, in this case, any part with a triangle inside the box or intersecting the box will be elected. Upon creation a FilterAABBInterface is set to elect all parts with a loose policy, i.e. all parts having one triangle overlapping the box will be elected, and a unit box centered on (0,0,0).

1.4 - Filter part instance list

A filter that knows the parts that should be included in the filtering explicitly. The parts are known by their part instance ids (THIS IS NOT their geometric ids). Such a filter may be created by first determining the parts instances ids with a filter solver, asking to retrieve their part instance ids and then create a filter with the given ids.

1.5 - Filter Attribute

A filter to elect parts based on their attributes. This filter elects parts having a specific attribute whose value :

- is exactly.
- contains at least one of the values (full text search).

in a predetermined list.

1.6 - Filter Range

A filter to elect part based on their attributes. This filter elects parts having a specific attribute whose value is included inside a set of range.

1.7 - Filter Has Field

A filter to elect parts based on their attributes. This filter elects parts having a specific attribute whatever its values.

1.8 - Filter Boolean

A filter to elect parts based on their attributes. This filter elects parts having a specific attribute whose value is True or False.

1.9 - Filter Literal

The filter select parts from the result of the query. The query is written in the CQL language (see the native client application for more details).

1.10 - Filter set

The filter set is a filter container to gather filters together. Each filter inside a set filter is computed separately and combine with the others filter depending to it's operator.

1.11 - Filter Compound

The filter compound is a filter container, like a set filter, to filter further several criteria at the same node in the product structure scheme.

Search

SearchInterface is used to perform search requests in the DMU. Instanciates search interface with the metadata manager.

```
mMetadataManager = mEngineInterface.getMetadataManagerInterface();  
mSearch = mMetadataManager.createSearch();  
mSearch.addEventListener("ready",onSearchReady);
```

Triggers a search with [search](#) function. The query is written in the CQL language (see the native client application for more details). The user may :

- Set a visibility context to limit the search on the visible parts
- Set a conf context to limit the search to some configuration(s)
- Choose to discard parts matching the query and configuration but not matching the visibility (if not, part is present but mInVisibilityCtx is set to false)
- Limit the search to only p results by setting pMaxPartResult
- Customize the result of the search with pElasticSourceFilter

The pElasticSourceFilter parameter defines the fields to retrieve of the metadata part document. The parameter must be a string containing a valid JSON expression. If you return `"metadata.*"` (mind the surrounding quotes), all metadata keys will be available in the SearchPartResult. On the contrary, if it is empty, no metadata will be retrieved. Please refer to the [ElasticSearch](#) documentation on [source](#) filtering

(<https://www.elastic.co/guide/en/elasticsearch/reference/1.3/search-request-source-filtering.html>)ex: if `pElasticSourceFilter` is set with `["metadata.Name","metadata.srcfile"]`, the result will return the content of the field name and the field srcfile from the part metadata document.

An event *ready* is fired when the search is finished, then *getLastError* tells if the search was correctly performed. Gets the list of all geometric ids which matched the search with *getGeometricInstanceIds* function. Not limited to the `maxPartResult` count. Or gets the list of all search part results with *getSearchParts*. Limited to the `maxPartResult` count. A search part result is define by:

- The id of the part.
- The name of the part.
- The metadata document of the part. The content depends on the parameter `pElasticSourceFilter` set on the search request.
- Tells if the part is in the visibility context set in the search request.
- Tells if the document returned is from a link metadata

Materials

Allows coloring the dmU. This interface is used to create and set materials for geometric objects. Warning, materials cannot be removed. It is strongly advised to modify existing materials once not in use, rather than creating new materials. Moreover, there is a fixed limit for the total number of different materials that may be in use. Depending on the number of materials of the DMUs, there may be a different number of custom materials that may be created. The number of custom materials that may be created is given by 'getNbAvailableCustomMaterials'. Creates new materials with *createNewMaterial* function with parameter given the the new diffuse color (3 floats in the range [0:1]). To manipulates materials of geometric instance uses function *changeMaterialOfInstance*. To restore original materials use the function *restoreOriginalMaterialOfInstance*.

```
mMaterialManager = mEngineInterface.getMaterialManager();  
mMaterialId = mMaterialManager.createNewMaterial(new Vector3(1.,1.,0.)  
);  
mMaterialManager.changeMaterialOfInstance(instanceid, mMaterialId);  
...  
mMaterialManager.restoreOriginalMaterialOfInstance(instanceid);
```

Navigation

This section describe how to manipulate the camera, and thus to change the viewpoint. Retrieve the camera manager from the infinite engine interface, with the function `getCameraManager`. Then manipulates the camera location and orientation with:

- `setCameraLocation` sets the camera location.
- `setCameraFrame` sets the camera frame of the camera. Warning the three vector must be orthogonal between them.
- `moveTo` sets the position and direction of the camera.
- `resetCamera` resets the camera position to fit the entire visible scene.
- `lookat` sets the position of center of interest, the camera will turn to look at the point.
- `fitBox` and `fitGeometry` changes the camera position and orientation to fit the given AABB, or geometric instance id to the screen.

Changes the sensitivity of the control (ie the speed) with the `setControllerSensitivity` function. And sets the frame reference with `setFrameReference` function witch defines the front and the top of your scene.

Converter

In order to manipulate internal identifier, we provide some converter to translate ids. There are three different id used in the api:

- **Part id** is an identifier of a model or template, be it an assembly or a single element. This part is referenced by the digital mock-up but is not present (instanciated) within it. It is not localized in the digital mockup. Ex: a wheel.
- **Part instance id** is an identifier of one instance of a part, be it an assembly or a single element. This instance has a 3D representation located in the digital mock-up. Ex: the front-right wheel.
- **Geometric instance id** is an identifier of a 3D representation of a part. For performance reason, 3D Juump Infinite merge same 3D representation instanciated at the same location. It means one geometric instance id may represent one or more part instance ids.

1.1 - Part converter

To convert a part id, create a part converter object with the metadata manager interface. Then connect the converter to the event *ready* fired when the result is ready and to the event *cancelled* fired when the conversion is cancelled.

```

mMetadataManager = mEngineInterface.getMetadataManagerInterface();
mConverter = mMetadataManager.createPartConverter();
mConverter.addEventListener("ready", onPartConverterReady);
mConverter.addEventListener("cancelled", onPartConverterCancelled);

```

Then call the convert function to start an asynchronous conversion. If you specify the conf context and/or the visibility context, the result will be limited to the conf or visibility.

```

mConverter.convert( pPartId, pConfCtx, mVisibilityCtx);

```

On *ready* event callback you can retrieve the result of the conversion. The result is composed by a list of all geometric instance id corresponding to the given part id and a list of couple {part instance id, list of geometric instance id}.

```

let onPartConverterReady = (pEvent) => {
  let lPartConverter = pEvent.emitter;
  if(lPartConverter.isRunning()) {
    return;
  }

  if(lPartConverter.getLastError().length == 0) {
    //show the error
    return;
  }

  let lResult = lPartConverter.getPartConversionResult();
  if(!lResult){
    return;
  }
  let lPartConverterInstances = lResult.getPartConverterInstances();
  let lAllGeometricInstanceIds = lResult.getAllGeometricInstanceIds(
);
  ...
}

```

1.2 - Part instance converter

To convert part instance id, create a part instance converter object with the metadata manager interface. Then connect the converter to the event *ready* fired when the result is ready and to the event *cancelled* fired when the conversion is cancelled.

```

mMetadataManager = mEngineInterface.getMetadataManagerInterface();
mConverter = mMetadataManager.createPartInstanceConverter();
mConverter.addEventListener("ready", onConverterReady);
mConverter.addEventListener("cancelled", onConverterCancelled);

```

Then call the convert function to start an asynchronous translation from part instance id list to the corresponding geometric instance ids.

```

mConverter.convert( pPartInstanceIds);

```

On *ready* event callback you can retrieve the result of the conversion. The result is composed by a list of geometric instance id corresponding to the given part instance ids.

```

let onPartConverterReady = (pEvent) => {
  let lPartInstanceConverter = pEvent.emitter;
  if (lPartInstanceConverter.isRunning()) {
    return;
  }

  if (lPartInstanceConverter.getLastError().length !== 0) {
    return;
  }

  let lGeometricIntanceIds = lPartInstanceConverter.getGeometricInst
anceIds();
  if (lGeometricIntanceIds){
    ...
  }
}

```

1.3 - Geometric instance converter

To convert geometric instance ids, create a geometric instance converter object with the metadata manager interface. Then connect the converter to the event *ready* fired when the result is ready and to the event *cancelled* fired when the conversion is cancelled.

```

mMetadataManager = mEngineInterface.getMetadataManagerInterface();
mConverter = mMetadataManager.createGeometricInstanceConverter();
mConverter.addListener("ready", onConverterReady);
mConverter.addListener("cancelled", onConverterCancelled);

```

Then call the convert function to start an asynchronous translation from list of part geometric instance id, to the corresponding part instance ids. The conf context and the VisibilityContext are optional. If the conf context is set the result will be limited to configuration. If the visibility is set the result will be limited to visibility. If the conf and the visibility is set the visibility context has priority.

```

mConverter.convert(pGeometricInstanceIds, pConfContext, pVisibilityCon
text)

```

On *ready* event you can retrieve the result of the conversion. The result is compose by a list of geometric instance id corresponding to the given part instance ids.

```

let onPartConverterReady = (pEvent: InfiniteEvent) => {
  let lPartInstanceConverter = pEvent.emitter;
  if (lPartInstanceConverter.isRunning()) {
    return;
  }

  if (lPartInstanceConverter.getLastError().length !== 0) {
    return;
  }

  let lGeometricIntanceIds = lPartInstanceConverter.getGeometricInst
anceIds();
  if (lGeometricIntanceIds){
    ...
  }
}

```

```
}  
}
```

Rendering primitives

Primitive manager interface is used to render primitive elements, such as points, lines and boxes. This interface also holds the materials that are used by primitive elements. Gets access to the different primitive with the specific manager *getPointManager*, *getLineManager*, *getBoxManager* and *getMaterialManager*.

1.1 - Point

Points are billboarded circular elements, with a pixel size, an inner color, an outer color and the size of the inner circle. On point creation with *createPoint* function, it is assigned a positive id that will represent this point from now on. 0 is an invalid point id. Please note that this id will be reused if the point is removed and another created again. Before creating a point, materials to render this point must be created in the primitive manager.

```
mPrimitiveManager = mEngineInterface.getPrimitiveManager();

//Create primitive materials
mMaterialManager = mPrimitiveMgr.getMaterialManager();
mRedMatId = mPrimitiveMatMgr.createMaterial(new Vector4(1.,0.,0.,1.));
mWhiteMatId = mPrimitiveMatMgr.createMaterial(new Vector4(1.,1.,1.,1.));
);

//Create a 3D primitive point
```



```

mPointManager = mPrimitiveMgr.getPointManager();
mPointId = mPointManager.createPoint(new Vector3(x,y,z), 12, 10,mRedMatId,mWhiteMatId);
...
// and remove it
mRemoved = mPointManager.removePoint( mPointId);

```

1.2 - Line

Creates a new line given two points positions, A and B, with a given width. The width of the line and the color of the line are set by the user. On line creation with `createLine` function, it is assigned a positive id that will represent this line from now on. 0 is an invalid line id. Please note that this id will be reused if the line is removed and another created again. Before creating a line, materials to render this point must be created in the primitive manager.

```

mPrimitiveManager = mEngineInterface.getPrimitiveManager();

//Create primitive materials
mMaterialManager = mPrimitiveMgr.getMaterialManager();
mWhiteMatId = mPrimitiveMatMgr.createMaterial(new Vector4(1.,1.,1.,1.));

//Create a 3D primitive line
mLineManager = mPrimitiveMgr.getLineManager();
mLineId = mLineManager.createLine( pointA, pointB, mWhiteMatId, 3);
...
// and remove it
mRemoved = mLineManager.removeLine( mLineId);

```

1.3 - Box

Creates a new box at the given position with the given size. A box is an axis aligned box, consisting of 12 lines, and 6 faces. The width of the lines, the color of the lines, the color of the faces are set by the user. On box creation with `createBox` function, it is assigned a positive id that will represent this box from now on. 0 is an invalid box id. Please note that this id will be reused if the box is removed and another created again. Before creating a box, materials to render this box must be created in the primitive manager.

```

mPrimitiveManager = mEngineInterface.getPrimitiveManager();

//Create primitive materials
mMaterialManager = mPrimitiveMgr.getMaterialManager();
mWhiteMatId = mPrimitiveMatMgr.createMaterial(new Vector4(1.,1.,1.,1.));
mRedMatId = mPrimitiveMatMgr.createMaterial(new Vector4(1.,0.,0.,1.));

//Create a 3D primitive box
mBoxManager = mPrimitiveMgr.getBoxManager();
mBoxId = mBoxManager.createBox( center, halfExtent, mWhiteMatId, mRedMatId, 3);
...
// and remove it
mRemoved = mBoxManager.removeBox(mBoxId);

```

1.4 - Primitive materials

Manager a primitive material to be used by primitive items. The color is expressed as a Vector4, RGBA (Red, Green, Blue, Alpha), Red = pRGBAColor.x, etc... Color values are expressed from 0 to 1. An alpha value of 1 means an opaque color, an alpha value of 0 means a transparent color. Materials are expressed and manipulated as positive ids. A material id of 0 is invalid. There is a limit on the number of materials that may be created. [*getNbAvailableCustomMaterials*](#) tells the number of materials that may be created. Materials cannot be removed, it is therefore advised to recycle old materials no longer in use and modify them.