[Prologue]

We can find mimeType.rdf datasource access code in **”/uriloader /exthandler/nsHandlerService.js”**.

The following are APIs(nsIHandlerService) implemented by nsHandlerService.js:

- nsISimpleEnumerator **enumerate()**;
- void **fillHandlerInfo**(in nsIHandlerInfo aHandlerInfo,
  in ACString aOverrideType);
- void **store**(in nsIHandlerInfo aHandlerInfo);
- boolean **exists**(in nsIHandlerInfo aHandlerInfo);
- void **remove**(in nsIHandlerInfo aHandlerInfo);
- ACString **getTypeFromExtension**(in ACString aFileExtension);

These APIs are composed of “**Retrieval Methods**” and “**Storage Methods**”.
Both “Retrieval Methods” and “Storage Methods” are using “**Datastore Utils**”[1] to access “mimeType.rdf”.

In order to get rid of mimeType.rdf, we need to rewrite “Datastore Utils” to access new format (currently JSON). We also need to make some modifications in “Retrieval Methods” and “Storage Methods” for the difference of two formats.

[Main]

[related bugs]

There are two related bugs mentioned in mail thread.
After analysis, I don't find the direct relation with RDF format.

Bug 332690 - ajaxSketch breaks handling of local SVG files
[comment] The root cause of the bug is the mime type of the corresponding extension is modified after clicking the specific link. The user **cannot** modify the mime type after the change. In my opinion, we can think this problem from UX spec. If the user can modify the value of mime type, the user won't stuck in this predicament.

Bug 167320 - eternal/endless/infinite loop when associating Firefox/SeaMonkey/Mozilla as a helper application for a given file type
[comment] I think we can solve this problem in the flow of content handling.

[Save data in JSON]

```
[RDF format]

<RDF:Seq RDF:about="urn:mimetypes:root">
  <RDF:li RDF:resource="urn:mimetype:application/pdf"/>
      ...
  <RDF:li RDF:resource="urn:mimetype:image/jpeg"/>
      ...
  <RDF:li RDF:resource="urn:mimetype:text/plain"/>
      ...
</RDF:Seq>

<RDF:Seq RDF:about="urn:schemes:root">
  <RDF:li RDF:resource="urn:scheme:webcal"/>
  <RDF:li RDF:resource="urn:scheme:ircs"/>
  <RDF:li RDF:resource="urn:scheme:mailto"/>
      ...
</RDF:Seq>
```

```
[JSON format]

{
  "mimetypes" : [
    {JSON of application/pdf},
        ...
    {JSON of image/jpeg},
        ...
    {JSON of text/plain},
        ...
  ],
  "schemes" : [
    {JSON of webcal},
    {JSON of ircs},
    {JSON of mailto},
        ...
  ]
}
```

```
[RDF format]

  <RDF:Description RDF:about="urn:mimetype:image/jpeg"
                   NC:value="image/jpeg"
                   NC:editable="true"
                   NC:fileExtensions="jpg"
                   NC:description="">
    <NC:handlerProp
    RDF:resource="urn:mimetype:handler:image/jpeg"/>
  </RDF:Description>

  <RDF:Description RDF:about="urn:mimetype:handler:image/jpeg"
                   NC:alwaysAsk="true"
                   NC:saveToDisk="true">
    <NC:externalApplication
    RDF:resource="urn:mimetype:externalApplication:image/jpeg"/
    >
  </RDF:Description>
```

```
[JSON of image/jpeg]

{
    "image/jpeg" : {
      "editable" : "true",
      "fileExtensions" : "jpg",
      "description" : "",
      "alwaysAsk" : "false",
      "saveToDisk" : "true"
    }
}
```

[Smooth Transition]

- A PREF for using JSON format.
- Develop the functionality of format conversion
    - [TODO Discussion] If we need to add a new API for FE usage.

[Flow]

1. Check the PREF first:
   If PREF is on, try to find mimeTypes.json(goto 2).
   If PREF is off, just do the same thing as before.
2. Check if there is mimeType.json:
   If mimeType.json exists, goto 3.
   If mimeType.json doesn't exist, do the conversion. Then 3.
3. We will use new "Datastore Utils" to access mimeType.json.

[Schedule]

- [1W. 7/8] File IO. (Write the in-memory DB to disk and read the data from the disk)
- [2W. 7/22] Rewrite all needed methods. ("Retrieval Methods","Storage Methods" and Datastore Utils)
- [1W. 7/29] Format conversion.
- [1W 8/5] Front-End integration and testing

[Epilogue]

[1] (Datastore Utils)

_getValue : Get the value of a property of an RDF source.
_hasValue : Whether or not a property of an RDF source has a value.
_getTargets : Get all targets for the property of an RDF source.
_setLiteral : Set a property of an RDF source to a literal value.
_setResource : Set a property of an RDF source to a resource target.
_setTarget : Assert an arc into the RDF datasource if there is no arc with the given source and property; otherwise, if there is already an existing arc, change it to point to the given target.
_addResourceAssertion:
_removeResourceAssertion:
_hasResourceAssertion:
_hasLiteralAssertion:
_existsLiteralTarget : Whether or not there is an RDF source that has the given property set to the given literal value.
_existsResourceTarget:
_removeTarget : Remove a property of an RDF source.
_removeAssertions : Remove all assertions about a given RDF source.

_getInfoID : return “urn”+ aClass + “:handler” + aType
_getTypeID : return “urn”+ aClass + “:” + aType
_getClass : return CLASS_MIMEINFO or CLASS_PROTOCOLINFO
_getPreferredHandlerID : return "urn:" + aClass + ":externalApplication:" + aType;
_getPossibleHandlerAppID : return handlerAPPID (local: or web: or dbus:)

_ensureAndGetTypeList : return typeList (RDF container)

_ensureRecordsForType: Make sure there are records in the datasource for the given content type by creating them if they don't already exist.
_appendHandlers: Append known handlers of the given class to the given array.