DesktopX Standards Document

Version 0.2

(Draft proposal)

By Martin Conroy

The Reason for this Standard

As DesktopX authors have become more sophisticated in their object and widget development they have taken to storing information relating to their creations in a range of ways.

The problem is that is that there is no consistency in the way things are done, even where the goal of the widget is the same (such as displaying the weather). As such there is a range of problems:

- The user experience is poor if they need to enter similar preferences for each different object
- The development of objects is made more complex as the developer needs to develop code for storing information
- A lack of information about the user makes more it difficult for developers to personalize widgets to the user
- There is no standard for the various types of common information used and as such bugs are more widespread and it's more difficult for users to learn how to develop their own objects.
- There is no standard for widgets to store other information outside of this common user information
- Transferability of information is poor when changing machines, reinstalling etc.

The Goals of this Standard

In order to address the above issues this document has several goals:

- Common User Information (CUI)
    - A xml document standard to store this information
    - A DesktopX Widget to provide a means to enter/update this information for the user
    - A series of scripts to assist in retrieval and manipulation this data
- Additional Object Information (AOI)
    - A xml document standard to store this information
    - A DesktopX Widget to provide a means to enter/update this information for the user
    - A series of scripts to assist in retrieval and manipulation this data
- Common Scripts
    - A core series of scripts to allow support developers in their widget creation. These objects will include (but not be limited to):
        - The above CUI manipulation scripts
        - The above AOI manipulation scripts
        - Registry retrieval e.g. (local timezone, language preferences)
        - XML manipulation
        - Common conversions/data manipulation
- Common object templates
    - A series of bases for commonly made objects including (but not limited to):
        - Weather
        - News
        - Calendar

Version control

The output of all the four goals will be stringently version controlled to ensure it is an effective solution.

All the scripts developed will need to incorporate version checking to ensure that they are based upon the latest standards.

For example, the Common User Information interface will need to check that it is the latest versions otherwise users may find that they download widgets that try to use CUI that they have not entered.

Developers can also implement solutions that will prompt users when a new version of their object is available.

Developments

Developments of the standards will need to be tightly controlled. This is crucial otherwise the point of the standards will be lost and development will be as anarchic as before.

That said, the standard is intended to be an open standard. Suggestions for its development are welcome. It is not intended that I will the final arbiter of what is included in the standard, though initially I will need to be fairly strict. I do however expect there to be a lot of additions in the short term

I will need to find a method by which standards can be proposed. I expect I will request a newsgroup be added. This will permit debate on proposals and be the most practical method by which I can assess interest and arguments.

The standards, especially for the CUI, will be an evolutionary process in terms of what is included but I want the syntax of the inclusion to be decided as early as possible to ensure that we ‘tweak’ the standard as little as possible.

Over time, should these standards be widely adopted, I would expect some form of panel to agree standards rather than myself.

Standard Approval

I think the key to getting this standard adopted widely is by support of some of the major DesktopX developers. I would hope that I can get the support of these developers to help with the adoption of standards by using it in the creation of new objects, and ideally by updating older objects.

In time I would like to see some form of “DX Standards approval” that could be applied to objects as an indicator of there adherence to standards. This will be something that can be considered in time depending on the success of this project.

Common User Information (CUI)

Purpose

The CUI is an XML file where information that is likely to be of common usage can be stored. It allows users to specify their details and a preference for a range of items and allows developers to use these in order to provide a more customized user experience.

Editing

In the near future a DesktopX Widget will be created to allow the user to edit the contents of the CUI file. There should be absolutely no need to edit the CUI file directly. This can only lead to errors being introduced.

Structure

This section outlines the structure of the CUI. It should be the primary source of reference for developers wishing to use the CUI. The common scripts will provide a mechanism for retrieving this information.

Basic framework

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- This document is intended to be used to allow DesktopX users to add
preferences that assist DesktopX developers in meeting their requirements  -->
- <dxCUI version="0.2">
  + <personal>
  + <environment>
  + <shortcuts>
  + <media>
  + <information>
  </dxCUI>
```

The basic framework of the CUI document is very simple. There is a single variable containing the version number, and the rest of the document is user information split into basic categories.

The version number will be used in script to manage version control as described earlier.

These data categories are intended to be generic and cover most requirements. Whilst there may be a few more categories to add here, the intention is that most items be categorized into these sections which will now be described in more detail.

In time it may be that scripts be developed to allow the user to choose to update their information based upon something the developer suggests. For example, if the developer were to create a newsfeed object that (in addition to the users preferred sources) provided some new sources of information it may be appropriate that the user can add these to their preferences much as you can add a webpage you like to your Favorites.

<personal>

Personal information is split into 3 distinct sections:

```
- <personal>
  + <name>
  + <address>
  + <contact>
  </personal>
```

<name>: For purposes of addressing the user by name

<address>: For purposes of locating the user and providing local information

<contact>: For purposes of allowing the user to provide their contact information

<name>

A user's name is split into component parts to allow the developer to address the user in any structure they wish. Obviously with this, as with any script, you cannot assume that any item of data will be present and as such have checks built into scripts to cater for lack of information.

```
- <name>
    <title>Mr</title>
    <forename>Martin</forename>
    <middlename>Paul</middlename>
    <surname>Conroy</surname>
    <suffix />
    <nick>_Martin_</nick>
  </name>
```

<title>: The user's title such as 'Mr', 'Miss', 'Mrs', 'Dr', 'Rev' etc. This should be used when addressing the user formally.

<forename>: The users forename. This should be used when communicating with the user unless they explicitly provide a <nick>. Ideally in this circumstance you should provide an option for the user to specify how they want to be addressed.

<middlename>: The user can provide any middle names or initials here that may be useful when addressing the user more formally.

<surname>: The user's title such as 'Sr.', 'Jr.', 'IV' etc. This should be used when addressing the user formally.

<address>

A user's address is important for identifying where a user is located, typically to provide them with relevant information. The address should **never** be used to contact the user without their explicit permission. Always remember that the user provides their information on the basis of trust and this should not be abused.

```
- <address>
    <number>1</number>
    <line1>Halifax Close</line1>
    <line2>Full Sutton</line2>
    <city>York</city>
    <region>East Riding of Yorkshire</region>
    <country>United Kingdom</country>
    <postalcode>YO41 1NU</postalcode>
  </address>
```

You will often find combinations of this information important to ensure the user is provided with the correct information and to avoid errors. For example, logic like that below may be appropriate when obtaining information from weather.com

```
If country = "United States" Then
      Location = postalcode
Else
      Location = city & ", " & country
End If
```

<number>: The number or name of the property where the user resides. This may be a number (e.g. '1' or '1a') or a name (e.g. 'Mill Cottage' or 'Flat 13') depending on their property.

<line1> and **<line2>**: These two lines are likely to be the street and possibly a local area where the user resides. These are most likely to be useful where precise locations are needing, for example in mapping and directions.

<city>: The main city where the user resides. This will be most useful for providing information that is local to the user.

<region>: This will be the state, county or region of some sort where the user lives. It will be appropriate for more generalized information, or where difference circumstances occur in different areas (e.g. special offers for products).

<country>: It is of paramount importance that as a developer you consider that not all users live in the United States. As such this field is very important so you can ensure that you provide information appropriate for the user or if you cannot provide information, then at least advise the user.

<postalcode>: This is the users ZIP or postal code and is sometimes the best way to provide localized information. It does depend on the country and you should use this in conjunction with the country information to establish where this value can help. The above example regarding weather information is a prime example of this.

<contact>

These are the ways that a user can be contacted. They should only be used following a check that the user is happy that you do so.

You may wish to do something like request a users email address in order to send then some information, but even if this data field exists, you should check it is OK to do so first.

```
- <contact>
    <email>martin@stardock.com</email>
    <altemail>martin@martinconroy.net</altemail>
    <msn>martin@martinconroy.net</msn>
    <yahoo />
    <aim />
    <icq />
  </contact>
```

<email> and **<altemail>**: These are the primary and secondary email addresses of the user. Typically a primary email address should be used, though you may wish to offer the user a choice of which email address to use if you can see that both have been provided.

<msn> or **<yahoo>** or **<aim>** or **<icq>**: These are identifiers for the range of instant messaging solutions that are available.

<environment>

Environment information is information relating to the user's working environment and their preferences for it. A lot of the information for this is likely to come from the registry as part of the common scripts, however there are certain elements that can be selected by the user as part of this section:

<visual>: These are the users visual preferences which allow a user to provide objects which match as closely as possible with the user's working environment.

<time>: These are the users preferences as to how time settings will be displayed.

```
- <environment>
  + <visual>
  + <time>
  </environment>
```

<visual>

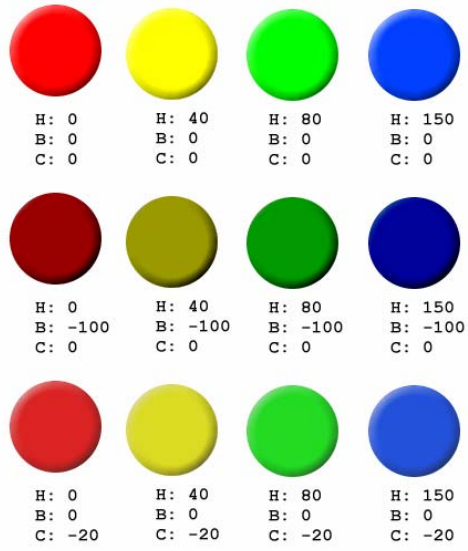
The main problem with a lot of objects that are created is that they are designed in such a way that they only fit in with a set color scheme.

These visual factors relate to a user's desktop that will help a developer make objects that coordinate and fit in with their environment. As a developer you must realize that the majority of users are not necessarily experienced enough to know how to tweak your objects so you must assist them in this.

```
- <visual>
    <hue>150</hue>
    <brightness>-100</brightness>
    <contrast>0</contrast>
    <textcolor>255,255,255</textcolor>
    <textcolor2>220,200,20</textcolor2>
  </visual>
```

<hue> and **<brightness>** and **<contrast>**: These are the standard DesktopX color controls that a user will manipulate until they find a combination that coordinates well with their desktop.

As a developer you should look at applying these to the primary areas of color in your object. All these settings are relative to an object that has a RGB value of 255,0,0. As such your base object before you apply these should be red. Here are some values that you can compare against.



<textcolor> and **<textcolor2>**: These are the user's preferred text colors to complement their desktop color. The <textcolor> parameter should be used for the majority of text in the object. The <textcolor2> parameter should be seen as a color to be used for highlighting or emphasis. For example this may be used for 'Mouse over' effects or to highlight areas with which the user may interact.

<time>

Different users have preferences as to how they like to see time elements displayed. Some are based on regional standards (e.g. date 'mm/dd/yyyy' vs. 'dd/mm/yyyy') whereas some are just personal (e.g. weekstart 'Sunday' vs. 'Monday'). As a developer you should look to support these preferences wherever possible. Even if the data you have (such as that returned from a website) does not meet these preferences you should aim to convert that data before displaying it.

<clockformat>: This specifies whether the user prefers to see time in a 12 hour format (4:25 PM) or a 24 hour format (16:25). The options are '12' or '24'.

<dateformat>: This specifies whether the user prefers to view dates in a 'dd/mm/yyyy' or 'mm/dd/yyyy' format.

<weekstart>: This specifies whether the week should be considered to start on 'Sunday' or 'Monday'. This is primarily of use in the creation of calendar objects.

<shortcuts>

This provides a mechanism for the user to provide links to programs and websites they find useful.

As a developer you cannot create an environment tailored to all users because they have their own preferred working environment.

As a developer you can create shortcuts to items like '.bmp' which will launch the default program for bitmap files. You can also create shortcuts to folders such as 'favorites'. These options however allow the user to specify the environment they want.

Within the <shortcuts> node the user can place an unlimited number of <sc> nodes to represent shortcuts. As a developer you can then manipulate these depending on the order in which they are presented or you can sort them based on something like their type.

```
- <shortcuts>
  - <sc>
      <name>WinCustomize</name>
      <link>http://www.wincustomize.com</link>
      <type>URL</type>
    </sc>
  + <sc>
  + <sc>
  + <sc>
  + <sc>
  </shortcuts>
```

<sc>

There are individual shortcuts and are defined with the following information:

<name>: This is the name of the application or the target of the shortcut.

<link>: This is the shortcut required to launch the application or the website.

<type>: This is the type of shortcut. By categorizing shortcuts the user allows to to provide context. For example you can group all URL type shortcuts together. The shortcut types are as follows:

- Folder
- URL
- Graphics
- Games
- Media
- Office
- Skinning
- Utility

<media>

This provides the user a way by which they can set some of their preferences regarding media players and media sources. In the current version the user can specify player preferences and also any radio stations that they prefer to listen to.

<players>

Within the <players> node the user can specify information for some of the most common media players that can be used to make DesktopX objects. Over time, if objects are made using additional players then these can be catered for. Each player is contained within its own <player> node.

```
- <players>
 - <player>
     <name>Windows Media Player</name>
     <volume>50</volume>
     <playlistdirectory>C:\Documents and Settings\Martin\My Documents\My Music\My Playlists</playlistdirectory>
     <playlist>C:\Documents and Settings\Martin\My Documents\My Music\My Playlists\All Music.wmp</playlist>
   </player>
 - <player>
     <name>iTunes</name>
     <volume>75</volume>
     <playlistdirectory />
     <playlist />
   </player>
 </players>
```

<name>: This is the name of the application. The currently supported players are:

- Windows Media Player
- iTunes

<volume>: This is the preferred default volume to which the player controls should be set when objects containing them are launched. You should also use this to store the volume of the player when the user exits the object.

<playlistdirectory>: This is the directory where the user stores custom playlists. As a developer you should be able to access the default directory but the user may specify an alternative here.

If the user has provided a directory here you should assume that they wish to use this rather than the default directory.

<playlist>: This is the default playlist that the user wants to load in their media player object. Whenever your object loads this is the playlist that should be prepared for playing.

<radio>

This allows the user to specify their preferred radio stations with instructions on how to connect to them.

Each station is specifies within a <station> node, and the user can specify as many of these as they like.

```
- <radio>
  - <station>
      <name>BBC World Service</name>
      <url>http://stream.servstream.com/ViewWeb/BBCTechnology/Event/BBC_World_Service.asx</url>
    </station>
  - <station>
      <name>Z95 Vancouver</name>
      <url>http://www.z95.com/player/resources/z95.com/live.asx</url>
    </station>
  </radio>
```

Within this node these are the following items specified by the user:

<name>: This is the name of the radio station

<url>: This is the web address on the online radio stream.

<information>

There is a wide range of sources of information that can be leveraged by DesktopX objects. Through use of the CUI, the user can specify their preferences for some of the more common of these.

This allows you as a developer to make an object of these types that are immediately tailored to the user specifications. As they are developed the ideal way to quickly develop this is through the Common Object Templates that are being developed to support these standards.

```
- <information>
  + <weather>
  + <rss>
  </information>
```

In the current version, weather and rss information feeds are supported.

<weather>

This provides the most common preferences for a weather object.

These are the units and locations.

```
- <weather>
    <units>Metric</units>
    <location>UKXX0162</location>
    <location>48188</location>
  </weather>
```

<units>: the type of units in which the data will be displayed. This is either “Metric” or “Imperial”.

<location>: the user can specify any number of locations for which weather information should be displayed. These relate to the specific location codes as used by weather.com. If preferences for these are not provided you should assume that you would use any <personal><address> information provided

<rss>

RSS feeds are a great way in which information is provided in a concise and structured way. The most common usage for this is in newsfeeds. As such the CUI is deigned to allow users to specify their preferred source of information.

Given that RSS feeds should follow a standard structure this allows developers to provide objects that are immediately tailored to use their preferred information source. As these feeds are typically dynamically generated then a developer can query then on a specified interval and then update information accordingly.

```
- <rss>
  - <feed>
      <name>BBC World News</name>
      <url>news.bbc.co.uk/rss/newsonline_ world_edition/front_page/rss091.xml</url>
      <type>News</type>
    </feed>
  + <feed>
  + <feed>
  + <feed>
  </rss>
```

The user can specify as many feeds as possible each containing 3 pieces of information:

<name>: The name of the information source.

<url>: The URL where the information can be found.

<type>: Information sources can be assigned to categories to allow the developer to provide information of a certain type if the user has made preferences. For example, a News object as opposed to a dedicated Sports object.

The current category types are as follows:

- News
- Sport
- Culture
- Entertainment
- Politics
- Technology
- Skinning

Additional Object Information (AOI)

Purpose

The AOI is an XML file where developers can store object specific information that is not present in the CUI. The AOI should **only** be used where the information required cannot be obtained from the CUI. The CUI should **always** be the first source of information if possible, and the AOI should not store equivalent data as this will totally devalue the purpose of standards.

Where an object can provide alternatives to elements stored in the CUI, such as working for alternative locations, the object should default to the user's location from the CUI. If an object wishes to allow any additional locations to be stored as "favorites" then these should be stored as part of the AOI, but even then, the object should provide an option to return to the default location.

Editing

A range of common scripts will be provided shortly that assist developers in writing data to this section.

Whilst to a degree, a developer has total freedom as to how they store data within the AOI it will be easier for the developer and all concerns if standards are adhered to.

The scripts provided should make it hard to justify going down any other route.

Structure

This section outlines the structure of the AOI. It is very simple and should be adhered to as closely as possible to ensure consistency.

Basic framework

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- This document is intended to be used to allow DesktopX developers to store preferences
in a structured manner aiding portability of settings and consistency of application   -->
- <dxAOI version="0.2">
  + <object>
  + <object>
  </dxAOI>
```

The basic framework of the AOI document is very simple. There is a single variable containing the version number, and the rest of the document is made up of <object> tags which reflect the individual objects placed their by developers.

When, as a developer you wish to add data to the AOI file, you will be adding your own <object> the first time a user runs your object. You will then update information within this tag as the user interacts with your object.

<object>

For your object you will specify 4 parameters and then you will place the objects information within a 5th <store> parameter.

<name>: The name of your object.

<author>: Your name. This along with the <name> will be used to uniquely identify the object, in order to access its data. As such it is crucial that you provide these fields. The standard scripts will assist you in this.

<email>: A contact address to allow users to speak to you if required.

<version>: This allows you to check the version of the data in the AOI. This could be of importance if you update your object and need to update information to a new format.

```
- <object>
    <name>Martin's Weather</name>
    <author>_Martin_</author>
    <email>martin@stardock.com</email>
    <version>1.0</version>
  + <store>
  </object>
- <object>
    <name>Martin's Notes</name>
    <author>_Martin_</author>
    <email>martin@stardock.com</email>
    <version>1.1</version>
  + <store>
  </object>
</dxAOI>
```

<store>

This is the area within which you have total control as a developer. Whilst you should stick to an xml style structure of parameters you have control of what parameters you use to store information. Obviously you should bear in mind the earlier guideline regarding not replicating CUI information.

In this example object based on would you would expect a Note-It! object to store you can see that the <store> contains everything needed in a simple clear to understand format.

The position of the pad is stored along with the number of notes. An array of individual <note> data is stored within the <notes> tag.

Another author may not choose to store the number of notes, but simply parse the <notes> array. As an author this is at your discretion.

```
- <object>
    <name>Martin's Notes</name>
    <author>_Martin_</author>
    <email>martin@stardock.com</email>
    <version>1.1</version>
  - <store>
      <notepadx>20</notepadx>
      <notepady>30</notepady>
      <notecount>2</notecount>
    - <notes>
      - <note>
          <message>Make new widget</message>
          <color>3</color>
          <x>150</x>
          <y>500</y>
        </note>
      - <note>
          <message>Have a rest</message>
          <color>2</color>
          <x>650</x>
          <y>125</y>
        </note>
      </notes>
    </store>
  </object>
```

Common Scripts

Purpose
TBC

Contents
TBC

Usage
TBC

Samples
TBC

Common Object Templates

Purpose
TBC

Contents
TBC

Usage
TBC

Samples