PowerToy DLL Project For Visual Studio 2022

Installation

- Put the `ModuleTemplate.zip` file inside the `%USERPROFILE%\Documents\Visual Studio 2022\Templates\ProjectTemplates\` folder, which is the default *User project templates location*. You can change that location via `Tools > Options > Projects and Solutions`.
- The template will be available in Visual Studio, when adding a new project, under the `Visual C++` tab.

Contributing

If you'd like to work on a PowerToy template, make required modifications to `\tools\project_template\ModuleTemplate.vcxproj` and then use the dedicated solution `PowerToyTemplate.sln` to export it as a template. Note that `ModuleTemplate.vcxproj` is actually a project template, therefore uncompilable, so we also have a dedicated `ModuleTemplateCompileTest.vcxproj` project referenced from the `PowerToys.sln` to help keeping the template sources up to date and verify it compiles correctly.

Create a new PowerToy Module

- Add the new PowerToy project to the `src\modules\` folder for all the relative paths to work.
- For the module interface implementation take a look at the interface.
- Each PowerToy is built as a DLL and in order to be loaded at run-time, the PowerToy's DLL name needs to be added to the `known_dlls` map in src/runner/main.cpp.

DPI Awareness

All PowerToy modules need to be DPI aware and calculate dimensions and positions of the UI elements using the Windows API for DPI awareness. The `/src/common` library has some helpers that you can use and extend:

- `dpi_aware.h`, `dpi_aware.cpp`
- `monitors.h`, `monitors.cpp`

PowerToy settings

Settings architecture overview

PowerToys provides a settings infrastructure to add a settings page for new modules. The PowerToys Settings application is accessed from the PowerToys tray icon, it provides a global settings page and a dedicated settings page for each module.

The PowerToys settings API provides a way to define the required information and controls for the module's settings page and methods to read and persist the settings values. A module may need a more complex way to configure the user's preferences, in that case it can provide its own custom settings editor that can be invoked from the module's settings page through a dedicated button.

The settings specification can be read at doc/specs/PowerToys-settings.md.

A PowerToy can provide this general information about itself:

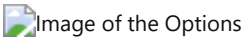
- **name**: The name of the PowerToy.
- **description**: A text describing the PowerToy.

- **icon_key**: The identifier of the PowerToy icon in the `settings-web` project.
- **overview_link**: A link to an extended overview of the PowerToy.
- **video_link**: A link to a video showcasing the PowerToy.

A PowerToy can define settings of the following types:

- **bool_toggle**: A boolean property, edited with a Toggle control.
- **int_spinner**: An integer property, edited with a Spinner control.
- **string**: A string property, edited with a TextBox control.
- **color_picker**: A color property, edited with a ColorPicker control.
- **custom_action**: A custom action property, invoked from the settings by a Button control.

Here's an example of what the settings look like in the Settings screen:

 Image of the Options

How to add your module's settings page

The PowerToy can set its settings information and controls by overriding the PowerToy's Interface `get_config` method and returning a serialized `PowerToysSettings::Settings` object that's been filled with the required information and controls.

The PowerToy can receive the new values by overriding the PowerToy's Interface `set_config` method, parsing the serialized `PowerToysSettings::PowerToyValues` object and applying the new settings.

Here's an example settings implementation:

```
  // Return JSON with the configuration options.
  virtual bool get_config(wchar_t* buffer, int* buffer_size) override {
    HINSTANCE hinstance = reinterpret_cast<HINSTANCE>(&__ImageBase);

    // Create a Settings object.
    PowerToysSettings::Settings settings(hinstance, get_name());
    settings.set_description(L"Serves as an example powertoy, with example
settings.");

    // Show an overview link in the Settings page
    settings.set_overview_link(L"https://github.com/microsoft/PowerToys");

    // Show a video link in the Settings page.
    settings.set_video_link(L"https://www.youtube.com/watch?v=d3LHo2yXKoY&t=21462");

    // Add a bool property with a toggle editor.
    settings.add_bool_toggle(
      L"test_bool_toggle", // property name.
      L"This is what a BoolToggle property looks like", // description or resource
id of the localized string.
      g_settings.test_bool_prop // property value.
    );

    // Add an integer property with a spinner editor.
    settings.add_int_spinner(
```

```
      L"test_int_spinner", // property name
      L"This is what a IntSpinner property looks like", // description or resource
id of the localized string.
      g_settings.test_int_prop, // property value.
      0, // min value.
      100, // max value.
      10 // incremental step.
    );

    // Add a string property with a textbox editor.
    settings.add_string(
      L"test_string_text", // property name.
      L"This is what a String property looks like", // description or resource id of
the localized string.
      g_settings.test_string_prop // property value.
    );

    // Add a string property with a color picker editor.
    settings.add_color_picker(
      L"test_color_picker", // property name.
      L"This is what a ColorPicker property looks like", // description or resource
id of the localized string.
      g_settings.test_color_prop // property value.
    );

    // Add a custom action property. When using this settings type, the
"PowertoyModuleIface::call_custom_action()"
    // method should be overridden as well.
    settings.add_custom_action(
      L"test_custom_action", // action name.
      L"This is what a CustomAction property looks like", // label above the field.
      L"Call a custom action", // button text.
      L"Press the button to call a custom action in the Example PowerToy" // display
values / extended info.
    );

    return settings.serialize_to_buffer(buffer, buffer_size);
  }

  // Called by the runner to pass the updated settings values as a serialized JSON.
  virtual void set_config(const wchar_t* config) override {
    try {
      // Parse the input JSON string.
      PowerToysSettings::PowerToyValues values =
        PowerToysSettings::PowerToyValues::from_json_string(config);

      // Update the bool property.
      if (values.is_bool_value(L"test_bool_toggle")) {
        g_settings.test_bool_prop = values.get_bool_value(L"test_bool_toggle");
      }

      // Update the int property.
```

```
      if (values.is_int_value(L"test_int_spinner")) {
        g_settings.test_int_prop = values.get_int_value(L"test_int_spinner");
      }

      // Update the string property.
      if (values.is_string_value(L"test_string_text")) {
        g_settings.test_string_prop = values.get_string_value(L"test_string_text");
      }

      // Update the color property.
      if (values.is_string_value(L"test_color_picker")) {
        g_settings.test_color_prop = values.get_string_value(L"test_color_picker");
      }

      // If you don't need to do any custom processing of the settings, proceed
      // to persists the values calling:
      values.save_to_settings_file();
      // Otherwise call a custom function to process the settings before saving them
to disk:
      // save_settings();
    }
    catch (std::exception ex) {
      // Improper JSON.
    }
  }
```

Settings Informations

The PowerToys settings object supports adding additional information to a PowerToys Settings description:

name

The name of the PowerToy. Its a required information that's applied in the settings object constructor:

```
PowerToysSettings::Settings settings(hinstance, get_name());
```

description

A short description of the PowerToy.

```
settings.set_description(L"Serves as an example powertoy, with example settings.");
```

or

```
settings.set_description(description_resource_id);
```

where `description_resource_id` is the UINT index of a resource string in the project .rc file.

icon_key

The identifier of the PowerToy icon in the `settings-web` project. By default, a `CircleRing` icon from *FabricUI* is shown for the PowerToy if no icon is specified.

```
settings.set_icon_key(L"pt-shortcut-guide");
```

overview_link

A link to an extended overview of the PowerToy.

```
settings.set_overview_link(L"https://github.com/microsoft/PowerToys");
```

video_link

A link to a video showcasing the PowerToy.

```
settings.set_video_link(L"https://www.youtube.com/watch?v=d3LHo2yXKoY&t=21462");
```

Setting Controls

bool_toggle

A boolean property, edited with a Toggle control.

It can be added to a `Settings` object by calling `add_bool_toggle`.

```
// Add a bool property with a toggle editor.
settings.add_bool_toggle(
  L"test_bool_toggle", // property name.
  L"This is what a BoolToggle property looks like", // description or resource id of
the localized string.
  g_settings.test_bool_prop // property value.
);
```

It can be read from a `PowerToyValues` object by calling `get_bool_value`.

```
// Update the bool property.
if (values.is_bool_value(L"test_bool_toggle")) {
  g_settings.test_bool_prop = values.get_bool_value(L"test_bool_toggle");
}
```

int_spinner

An integer property, edited with a Spinner control.

It can be added to a `Settings` object by calling `add_int_spinner`.

```
// Add an integer property with a spinner editor.
settings.add_int_spinner(
  L"test_int_spinner", // property name
  L"This is what a IntSpinner property looks like", // description or resource id of
```

```
the localized string.
  g_settings.test_int_prop, // property value.
  0, // min value.
  100, // max value.
  10 // incremental step.
);
```

It can be read from a `PowerToyValues` object by calling `get_int_value`.

```
// Update the int property.
if (values.is_int_value(L"test_int_spinner")) {
  g_settings.test_int_prop = values.get_int_value(L"test_int_spinner");
}
```

string

A string property, edited with a TextBox control.

It can be added to a `Settings` object by calling `add_string`.

```
// Add a string property with a textbox editor.
settings.add_string(
  L"test_string_text", // property name.
  L"This is what a String property looks like", // description or resource id of the
localized string.
  g_settings.test_string_prop // property value.
);
```

It can be read from a `PowerToyValues` object by calling `get_string_value`.

```
// Update the string property.
if (values.is_string_value(L"test_string_text")) {
  g_settings.test_string_prop = values.get_string_value(L"test_string_text");
}
```

color_picker

A color property, edited with a ColorPicker control. Its value is a string with the `'#RRGGBB'` format, with two hexadecimal digits for each color component.

It can be added to a `Settings` object by calling `add_color_picker`.

```
// Add a string property with a color picker editor.
settings.add_color_picker(
  L"test_color_picker", // property name.
  L"This is what a ColorPicker property looks like", // description or resource id
of the localized string.
  g_settings.test_color_prop // property value.
);
```

The `'#RRGGBB'`-format string can be read from a `PowerToyValues` object by calling `get_string_value`.

```
// Update the color property.
if (values.is_string_value(L"test_color_picker")) {
  g_settings.test_color_prop = values.get_string_value(L"test_color_picker");
}
```

custom_action

A custom action property, invoked from the settings by a Button control. This can be used to spawn a custom editor by the PowerToy.

It can be added to a `Settings` object by calling `add_custom_action`.

```
// Add a custom action property. When using this settings type, the
"PowertoyModuleIface::call_custom_action()"
// method should be overridden as well.
settings.add_custom_action(
  L"test_custom_action", // action name.
  L"This is what a CustomAction property looks like", // label above the field: a
string literal or a resource id
  L"Call a custom action", // button text: a string literal or a resource id
  L"Press the button to call a custom action in the Example PowerToy" // display
values / extended info: a string literal or a resource id
);
```

When the custom action button is pressed, the PowerToy's `call_custom_action()` is called with a serialized `PowerToysSettings::CustomActionObject` object.

```
  // Signal from the Settings editor to call a custom action.
  // This can be used to spawn more complex editors.
  virtual void call_custom_action(const wchar_t* action) override {
    static UINT custom_action_num_calls = 0;
    try {
      // Parse the action values, including name.
      PowerToysSettings::CustomActionObject action_object =
        PowerToysSettings::CustomActionObject::from_json_string(action);

      if (action_object.get_name() == L"test_custom_action") {
        // Custom action code to increase and show a counter.
        ++custom_action_num_calls;
        std::wstring msg(L"I have been called ");
        msg += std::to_wstring(custom_action_num_calls);
        msg += L" time(s).";
        MessageBox(NULL, msg.c_str(), L"Custom action call.", MB_OK | MB_TOPMOST);
      }
    }
    catch (std::exception ex) {
      // Improper JSON.
    }
  }
```

Settings Persistence

By default, the PowerToys settings are persisted in the User's `%LocalAppData%\Microsoft\PowerToys` path. Each PowerToy has its own folder for saving the persisted settings data.

Loading and saving the settings in the default location can be achieved through the use of a `PowerToysSettings::PowerToyValues` object.

Loading settings

The PowerToy can load the saved `PowerToyValues` object through the use of the `load_from_settings_file` method.

Here's an example:

```
// Load the settings file.
void ExamplePowertoy::init_settings() {
  try {
    // Load and parse the settings file for this PowerToy.
    PowerToysSettings::PowerToyValues settings =
      PowerToysSettings::PowerToyValues::load_from_settings_file(get_name());

    // Load the bool property.
    if (settings.is_bool_value(L"test_bool_toggle")) {
      g_settings.test_bool_prop = settings.get_bool_value(L"test_bool_toggle");
    }

    // Load the int property.
    if (settings.is_int_value(L"test_int_spinner")) {
      g_settings.test_int_prop = settings.get_int_value(L"test_int_spinner");
    }

    // Load the string property.
    if (settings.is_string_value(L"test_string_text")) {
      g_settings.test_string_prop = settings.get_string_value(L"test_string_text");
    }

    // Load the color property.
    if (settings.is_string_value(L"test_color_picker")) {
      g_settings.test_color_prop = settings.get_string_value(L"test_color_picker");
    }
  }
  catch (std::exception ex) {
    // Error while loading from the settings file. Let default values stay as they
are.
  }
}
```

Saving settings

The PowerToy can save the `PowerToyValues` object received in `set_config` through the use of the `save_to_settings_file` method.

Here's an example:

```
// Called by the runner to pass the updated settings values as a serialized JSON.
virtual void set_config(const wchar_t* config) override {
  try {
    // Parse the input JSON string.
    PowerToysSettings::PowerToyValues values =
      PowerToysSettings::PowerToyValues::from_json_string(config);
...
    values.save_to_settings_file();
  }
  catch (std::exception ex) {
    // Improper JSON.
  }
}
```

Alternatively, the `PowerToyValues` object can be built manually and then saved if more complex logic is needed:

```
// This method of saving the module settings is only required if you need to do any
// custom processing of the settings before saving them to disk.
void ExamplePowertoy::save_settings() {
  try {
    // Create a PowerToyValues object for this PowerToy
    PowerToysSettings::PowerToyValues values(get_name());

    // Save the bool property.
    values.add_property(
      L"test_bool_toggle", // property name
      g_settings.test_bool_prop // property value
    );

    // Save the int property.
    values.add_property(
      L"test_int_spinner", // property name
      g_settings.test_int_prop // property value
    );

    // Save the string property.
    values.add_property(
      L"test_string_text", // property name
      g_settings.test_string_prop // property value
    );

    // Save the color property.
    values.add_property(
      L"test_color_picker", // property name
      g_settings.test_color_prop // property value
    );
```

```
    // Save the PowerToyValues JSON to the power toy settings file.
    values.save_to_settings_file();
  }
  catch (std::exception ex) {
    // Couldn't save the settings.
  }
}
```

Add a new PowerToy to the Installer

In the `installer` folder, open the `PowerToysSetup.sln` solution. Under the `PowerToysSetup` project, edit `Product.wxs`. You will need to add a component for your module DLL. Search for `Module_ShortcutGuide` to see where to add the component declaration and where to reference that declaration so the DLL is added to the installer. Each component requires a newly generated GUID (you can use the Visual Studio integrated tool to generate one). Repeat the process for each extra file your PowerToy module requires. If your PowerToy comes with a subfolder containing for example images, follow the example of the `PowerToysSvgs` component.