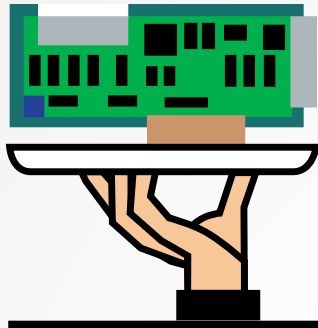
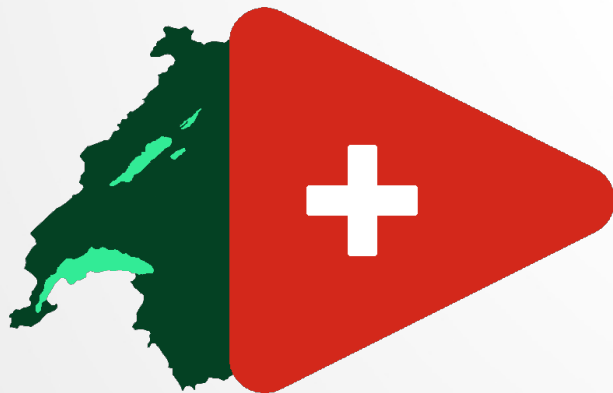




Sci-Consulting

présente

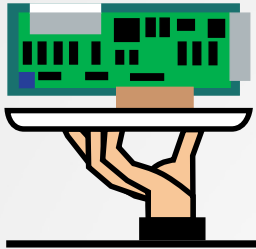
VI Malléables Workshop «hands-on»



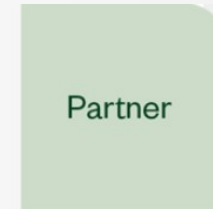
Gabriel Safar

Romandie LabVIEW User Group Meeting
16 juin 2025

Qui sommes-nous ?



Sci-Consulting



Services et solutions de développements sur mesure pendant 34 ans

Route Cantonale 100

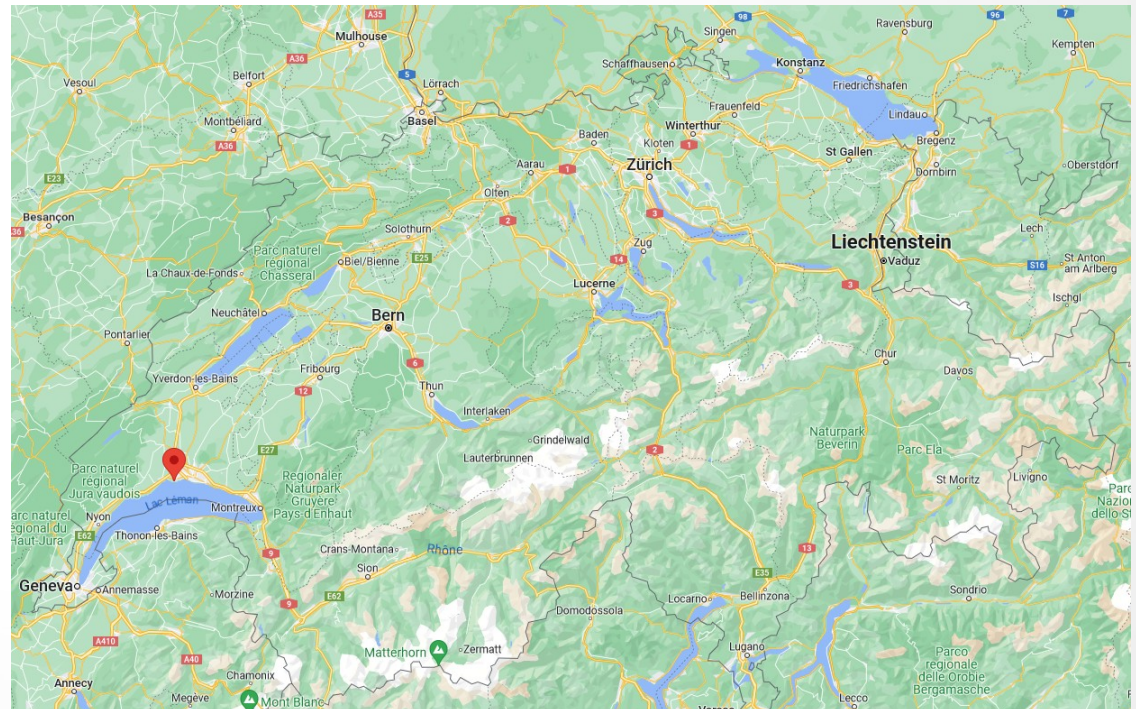
1024 Ecublens VD

Switzerland

021 697 07 61

<https://www.sci-consulting.ch>

info@sci-consulting.ch



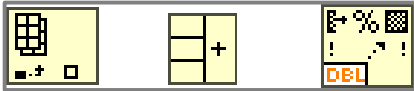
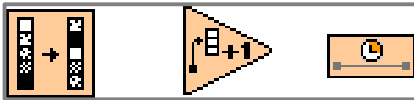
Sci-Consulting

16 juin 2025

Romandie LabVIEW User Group

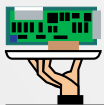
2

Qu'est-ce qu'un VI Malléable (VIM)?

- Un VI «inline» avec des terminaux d'entrée qui peuvent s'adapter au type de donnée du fil connecté
- Équivalent au polymorphisme dans l'orienté objet, mais sur les entrées du VI définis statiquement au compile time
- Fonctionnalité similaire à certaines fonctions de base de LabVIEW (blocs jaune)

- Certaines fonctions dans LabVIEW sont des VIM (blocs orange)

- Exemples dans LabVIEW (2017 au minimum):
`labview\examples\Malleable VIs\Basics\Malleable VIs Basics.lvproj`

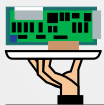
Utilisation des VIMs

- Exemples principaux d'entrée polymorphe utile:
 - String + Enum pour ce qui est descriptif
 - Scalaire + Array pour permettre le choix d'un élément ou plusieurs
 - Numérique ou Array de Numérique ou Waveform de Numérique avec différentes représentations (I32, U64, DBL, EXT, etc.)
 - Liste finie de cas spécifiques pour simplifier le code et garder la complexité dans un seul VI(M)
 - Générique car le type n'est pas déterminant pour la fonction elle-même (e.g. Array, Select, etc.)
 - Classes



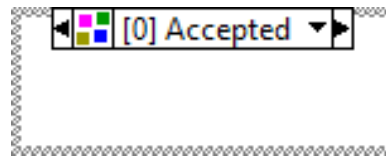
Forces et faiblesses

- Forces
 - Moins de duplication de code, plus de réutilisation de code
 - Fonctionnalités embarquées
 - Code défini au compile time, pas au run time
- Faiblesses
 - Le code peut être plus complexe qu'un VI standard
 - Pas debuggable en tant que tel (VI inline sans debugging)
 - Le Connecteur Pane doit rester fixe (vs VIs polymorphes)

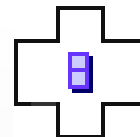
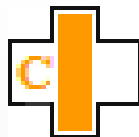


Outils pour coder des VIMs

- Outils dans la palette Comparison → Assert Type
 - Type Specialization Structure : défini les différents cas que le VIM doit essayer de compiler avec succès avec les entrées connectées, dans l'ordre croissant

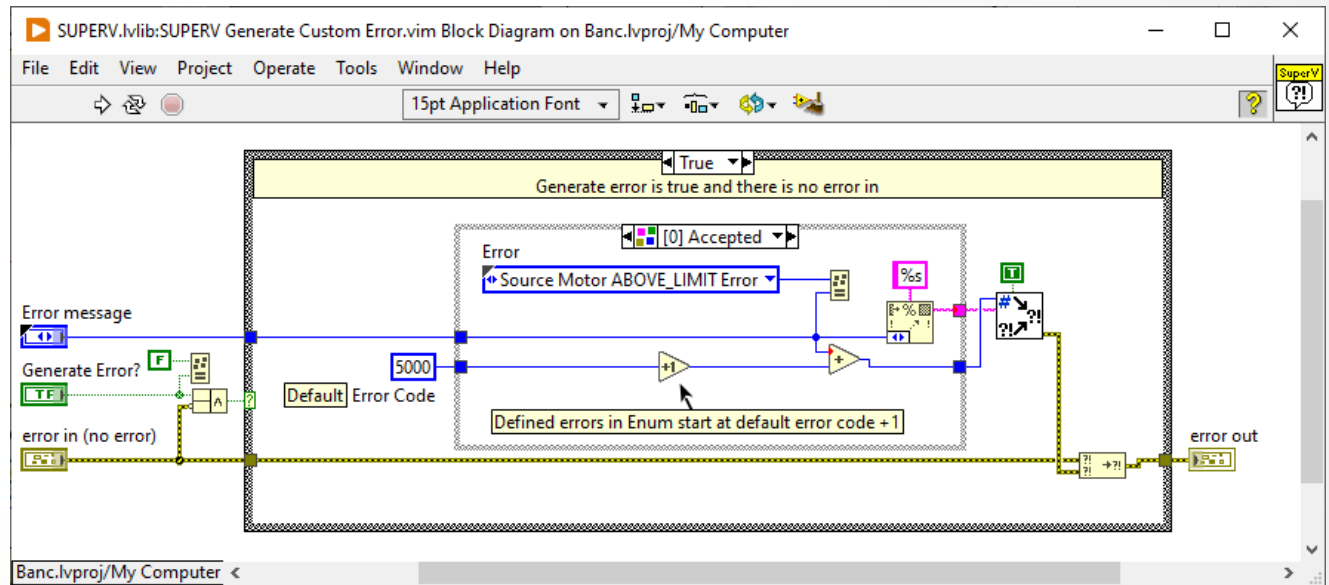
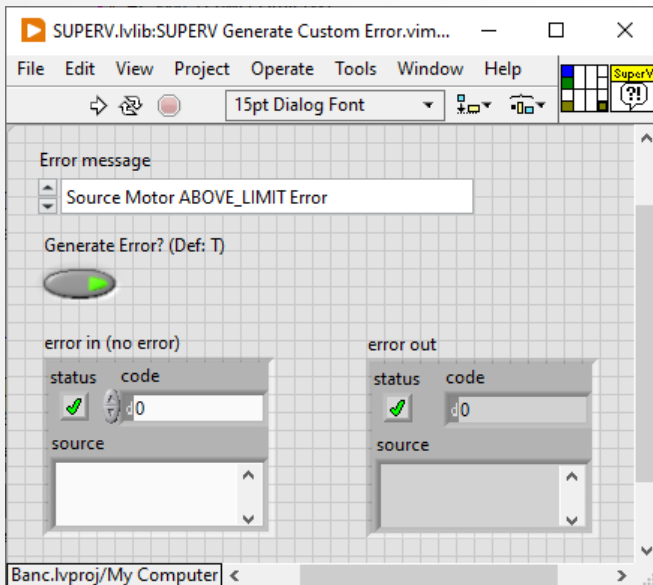


- Assert Structural Type Match / Mismatch: si les deux entrées sont du même / différents type(s), alors le bloc compile, sinon le VI est brisé  
- Et d'autres ! Vérification de type, de taille d'Array, ... et les fonctions que vous souhaitez utiliser !



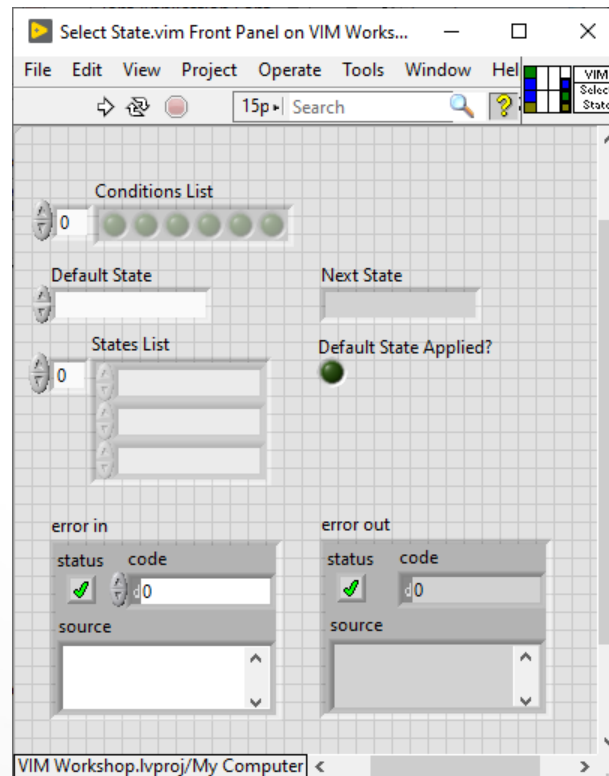
Hands-on – Exercice 1

- Coder un VIM qui va permettre de générer et merger une erreur custom à partir d'une entrée String ou Enum et d'une entrée booléenne



Hands-on – Exercice 2

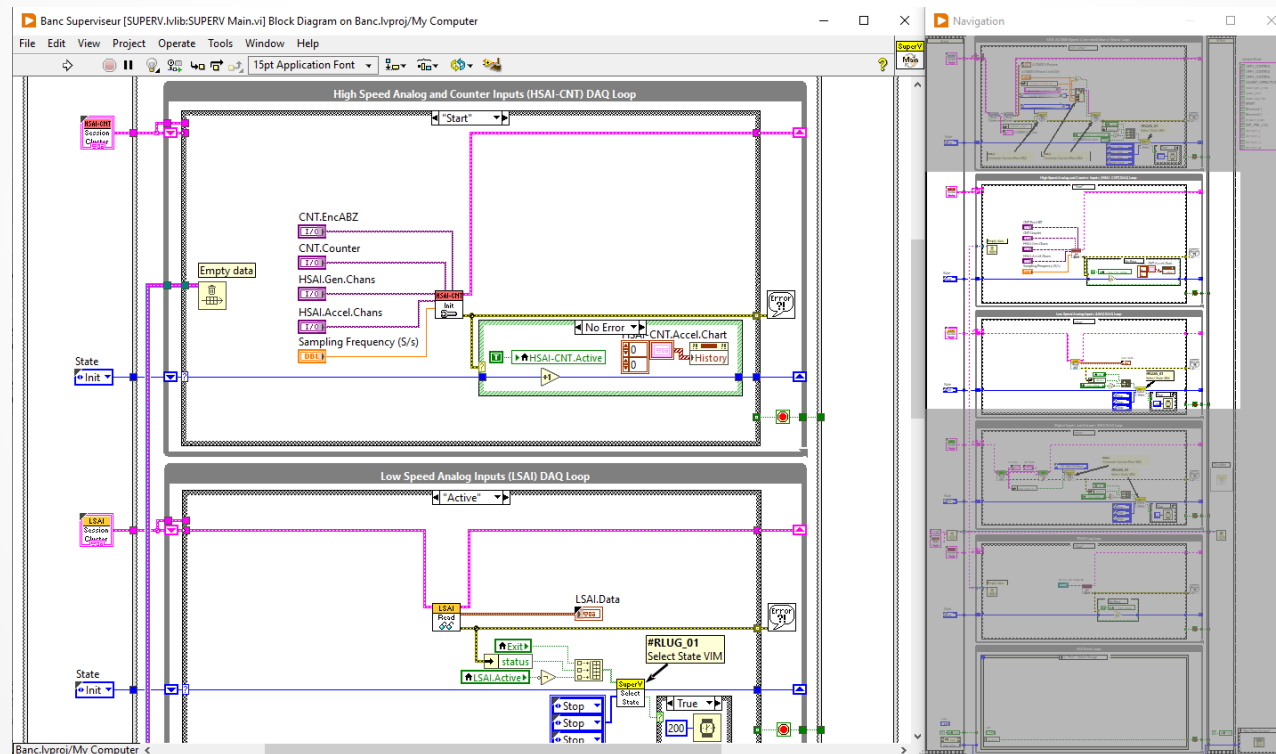
- Coder un VIM qui va permettre de sélectionner le prochain état d'une machine d'état parmi un array d'état en fonction d'un array de booléens, ainsi qu'un cas par défaut



Hands-on – Exercice 3

- Chercher dans un code existant du code qui peut être modifié ou transformé en VIM pour simplifier et modulariser... et les coder ! Download projet :

<https://www.sci-consulting.ch/download/RLUG/20250616/WS-Ex3.zip>



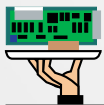
D'autres exemples tirés d'applications réelles

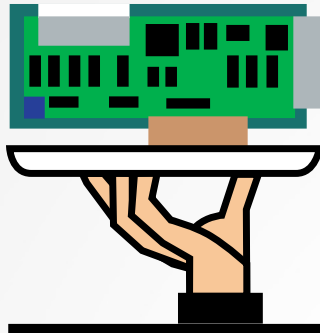
- **Set Waveform Channel Name** : String ↔ Enum
- **Add Prefix to Waveform Channel Names** : Wfm ↔ Array de Wfm
- **Convert Timestamp to and from String** : String ↔ Timestamp
- **Scale Value** : différentes représentations de Numeric
- **Convert Array String Data to Array Numeric Data** : différentes représentations de Numeric
- **Decimate Data** : Array de Numeric ↔ Wfm ↔ Array de Wfm
- **Convert Static Data to Value String** : String 1 & 2 (avec l'aide des variants!) ↔ Integer Numeric ↔ Boolean ↔ DBL Numeric
- **Replace or Insert** : Array de différents type d'éléments
- **Get Cluster Field Names** : N'importe quel Cluster avec des éléments qui ne sont pas des Clusters (profondeur = 1, avec l'aide des variants)
- **Index Contents List** : Array de différents type d'éléments
- Download exemples : <https://www.sci-consulting.ch/download/RLUG/20250616/WS-Exemples.zip>



Conclusion

- Utilisez les VIMs pour rendre certains de vos VIs encore plus modulaire avec des fonctionnalités embarquées
- Si vous devez dupliquer du code ou des VIs avec peu de changements entre eux, pensez d'abord à voir si un VIM pourrait faire l'affaire !
- Attention toutefois à la complexité, qui n'en vaut pas toujours la peine





Sci-Consulting

Merci de votre attention !

Des questions ?

→ info@sci-consulting.ch