41-écrire en JavaScript

introduction

Avec l'objet Max *js*, il est possible d'utiliser du code JavaScript pour exécuter des scripts de patcheur, où vous pouvez créer des objets Max dans un patcheur de manière dynamique, en définissant leurs propriétés, en leur envoyant des messages et en établissant des connexions entre eux. JavaScript vous permet d'utiliser du code de procédure pour générer des éléments de patcher d'une manière qui peut être plus difficile à faire par le biais de messages à l'objet *thispatcher* (l'autre manière de créer automatiquement des objets Max dans un patcheur). Ce didacticiel explique comment créer et supprimer des objets et des connexions dans un patch Max par le biais de méthodes personnalisées écrites en JavaScript, et montre également comment utiliser des méthodes pour traiter les messages personnalisés provenant du patcheur.

Pour ouvrir le patch du didacticiel, cliquez sur le bouton vert **Ouvrir didacticiel** dans le coin supérieur droit de la fenêtre de documentation.

Script du patcheur avec JavaScript

Lorsque vous ouvrez le patch du didacticiel pour la première fois, vous verrez un patcheur en grande partie vide avec un objet *js* dans la partie inférieure de la fenêtre du patcheur. L'objet *js* a chargé un fichier source JavaScript appelé 'autosurface.js', situé dans le même dossier que le patch du didacticiel.

L'objet *js* est configuré pour envoyer des nombres à un périphérique de sortie MIDI (à l'aide des objets *makenote* et *noteout*). Il possède également une sortie droite qui envoie des valeurs à l'entrée droite de l'objet *pack* qui envoie des messages à un objet *multislider*. De plus, notre objet *js* a un certain nombre d'objets connectés à son entrée. Un objet *metro* est connecté à notre objet *js*, tout comme deux boîtes de *messages* qui enverront les de messages **sliders $ 1** et **reverse $ 1**, où **$ 1** dans chaque cas est la valeur présente dans la boîte de *nombres* qui leur est connectée.

D'après la disposition du patch, nous pouvons en déduire que le code JavaScript de notre objet *js* devrait avoir au moins trois fonctions: **bang**, **sliders** et **reverse**. En fait, il en a un de plus, ce qui deviendra évident lorsque nous utiliserons le patch.

Auto-génération de patchs

Sélectionnez la boîte de *nombres* attachée à la boîte de *message* contenant le message **curseur $ 1**. Tapez ou faites défiler jusqu'au chiffre **5** et observez ce qui se passe. Changer la valeur dans la boîte de *nombres*. Essayez de la régler sur un nombre élevé (comme **50**).

Réglez-le sur **0** et voyez ce qui se passe.

En réponse au message **curseur $ 1**, notre objet *js* crée dynamiquement des objets et des connexions par le biais de scripts. Il crée des paires d'objets *ctlin* et *slider* pour correspondre au nombre de sliders que vous demandez par le message à l'objet *js*. De plus, il crée un objet *funnel* avec le nombre approprié d'entrées pour les objets *sliders* et établit les connexions appropriées entre eux. L'objet *funnel* est ensuite connecté à notre objet *js*, ce qui permet aux valeurs générées par les curseurs d'être également utilisées par notre code JavaScript.

Lorsque vous créez des curseurs, notez que les objets *ctlin* sont automatiquement numérotés pour écouter les numéros de contrôleur MIDI incrémentés. En conséquence, une surface de contrôle MIDI qui envoie des valeurs de contrôle continu MIDI sur plusieurs numéros de contrôleur envoie des valeurs à des objets *sliders* indépendants. Notez également que lorsque vous décrémentez le nombre de curseurs, les objets en surplus disparaissent (en fait, tout disparaît et est recréé à nouveau). Si vous fixez le nombre de curseurs à **0**, tous les objets créés par un script (y compris *funnel*) seront supprimés du patch.

Réglez le nombre de curseurs sur une valeur modeste, tel que **5**. Modifiez les valeurs des objets *sliders*, soit en cliquant dessus, soit en utilisant l'entrée d'un contrôleur MIDI. Activez l'objet *metro* en cliquant sur le *toggle* qui y est associé. Les valeurs des objets *sliders* doivent sortir de l'objet *js* à tour de rôle, créant une séquence de notes MIDI. Double-cliquez sur l'objet *noteout* pour sélectionner un synthétiseur MIDI valide. Vous devriez les entendre.

L'objet *multislider* situé à droite du patch vous donnera un affichage courant de la note actuelle sortant de notre séquenceur, placée à la position appropriée dans la séquence.

Cliquez sur le *toggle* attaché à la boîte de *message* contenant le message **reverse $ 1**. Notez que l'ordre dans lequel les valeurs du *slider* sont séquencées est maintenant inversé. Notre affichage *multislider* fonctionne également à l'envers.

En bref, notre objet *js* crée dynamiquement une surface de contrôle MIDI évolutive (avec des objets *ctlin* et *slider*) et utilise les valeurs de ces objets pour créer un simple ***step sequencer*** MIDI. Le nombre de curseurs créés par notre code JavaScript détermine la longueur de la séquence.

Désactivez les deux objets *toggle*, ce qui arrête la séquence et remet le transport du séquenceur en mode «forward». Regardons le code de notre objet *js*.

Le bloc global: tableaux et maxobjs

Double-cliquez sur l'objet *js* dans le patch du didacticiel. Le code pour 'autosurface.js' devrait apparaître. En haut du code devrait se trouver le bloc de commentaires habituel, expliquant ce que fait le script. Ci-dessous, nous devrions voir nos déclarations de code globales:

```
// entrées et sorties
inlets = 1;
outlets = 2;
// variables globales et tableaux
var numsliders = 0;
var seqcounter = 0;
var thereverse = 0;
var thevalues = new Array (128);
// Variables Maxobj pour le script
var controlin = new Array (128);
var thesliders = new Array (128);
var thefunnel;
```

Comme nous l'avons vu dans le tutoriel précédent, nos ***entrées*** et ***sorties*** en haut du code indiquent à *js* combien d'entrées et de sorties nous voulons dans notre objet.

Le bloc de code suivant définit certaines variables que notre code JavaScript devra utiliser de manière globale. Ces variables incluent: **numsliders**: stocke le nombre de 'curseurs' (paires de *ctlin* et de curseurs) que nous avons dans notre patch. Ceci est défini par le message *sliders* à notre objet *js*. **seqcounter**: stocke le point actuel dans notre séquence. Il est piloté par l'objet *metro* dans notre patch, et est donc modifié par un **bang ()** dans notre code. **thereverse**: indique si notre séquenceur fonctionne à l'envers ou non. Ceci est défini par le message **reverse** de notre objet *js*. **thevalues**: un tableau (voir ci-dessous) de valeurs reflétant l'état des objets *sliders* de notre patch. L'objet *funnel* de notre patch définit ces valeurs en envoyant des listes à notre objet.

Le nouveau constructeur **Array ()** crée des tableaux en JavaScript. La variable de tableau **thevalues**, ci-dessus, possède 128 éléments, auxquels on accède par la notation entre crochets après le nom du tableau, par exemple:

k = **thevalues** [*5*];

donnera à la variable **k** à la valeur du sixième élément (à partir de **0**) du tableau (array) **thevalues**.

thevalues [n] = *55*;

définira l'élément **n** de notre tableau **thevalues** à **55**.

Notez que JavaScript traite les tableaux en tant qu'objets, de sorte que:
k = thevalues.length;

fixera la variable **k** comme le ***nombre d'éléments*** du tableau **thevalues**. Pour plus d'informations à ce sujet, consultez toute bonne référence JavaScript.

Après nos déclarations de variable, nous avons des variables que nous utiliserons pour référencer des objets créés dynamiquement dans notre patch Max. Ces noms de variables sont utilisés en interne dans notre code JavaScript afin que nous puissions créer, connecter, supprimer et modifier des objets par le biais des propriétés associées à ces objets. Les objets *js* qui font référence aux objets Max dans un patcheur sont appelés ***Maxobjs***. Notre script contient les variables Maxobj suivantes: **controlin**: un tableau de Maxobj qui font référence aux objets ctlin de notre patch. thesliders: un tableau de Maxobjs qui font référence aux objets *slider* de notre patch. **thefunnel**: un Maxobj qui fait référence à l'objet *funnel* dans notre patch.

Notez qu'il n'y a pas de différence dans la déclaration de variable JavaScript par rapport au type de valeur que la variable stocke; les entiers, les flottants, les chaînes et les objets comme équivalents lors de la déclaration d'une variable. De même, les tableaux sont définis simplement pour faire référence à la quantité d'informations plutôt qu'au type d'informations qui y seront stockées. De même, JavaScript va taper correctement les variables après un calcul, par exemple:
x = *4/2*;

définira la variable **x** sur **2** (un entier), alors que:
x = *3/2*;

définira la variable **x** sur **1.5** (une valeur à virgule flottante). Les variables peuvent changer de type dynamiquement tout au long de leur existence. Cependant, cette utilisation de variables non typées n'existe que dans l'environnement JavaScript. C'est pourquoi nous avons toujours besoin de méthodes indépendantes (**msg_int ()** et **msg_float ()**) pour traiter les nombres typés différemment provenant de Max.

Nous allons utiliser différentes propriétés de la classe d'objets Maxobj pour effectuer notre script, le tout étant accompli par une seule fonction: notre méthode **sliders ()**.

Arguments, accords…

Notre objet *js* répond au message **sliders** via une méthode contenue dans la fonction **sliders ()** (rappelez-vous que le nom de la fonction correspond généralement au message que vous souhaitez voir déclencher pour cette fonction). Examinez le code de la fonction **sliders ()**. Les commentaires en haut de chaque section expliquent ce qui se passe à chaque étape du processus:

```
// sliders - génère et lie les curseurs dans le patch max
fonction sliders (val)
{
  if (arguments.length) // bail si aucun argument
  {
    // analyser les arguments
    a = arguments [0];
    // contrôle de sécurité pour le nombre de curseurs
    si (a < 0) a = 0; // trop peu de curseurs, mis à 0
    si (a > 128) a = 128; // trop de curseurs réglés sur 128
    // sort avec le vieux ...
    if (numsliders) this.patcher.remove (thefunnel); // si nous l'avons déjà fait,
     for (i = 0; i <numsliders; i ++) // supprime les objets ctlin et slider en utilisant l'ancien nombre de sliders
    {
      this.patcher.remove (controlin [i]);
      this.patcher.remove (thesliders [i]);
    }

    // ... avec le nouveau
    numsliders = a; // met à jour notre nombre global de curseurs à la nouvelle valeur
    if (numsliders) thefunnel = this.patcher.newdefault (300, 300, "funnel", a); // faire funnel
     for (var k = 0; k <a; k ++) // créer les nouveaux objets ctlin et uslider, les connecter les uns aux autres et à funnel
    {
      controlin [k] = this.patcher.newdefault (300+ (k * 100), 50, "ctlin", k + 1);
      thesliders [k] = this.patcher.newdefault (300+ (k * 100), 100, "uslider");
      this.patcher.connect (controlin [k], 0, thesliders [k], 0);
      this.patcher.connect (thesliders [k], 0, thefunnel, k);
    }

    // connecte de nouveaux objets à l'entrée de cet objet js
    ourself = this.box; // assigner un Maxobj à notre objet js
    if (numsliders) this.patcher.connect (thefunnel, 0, ourself, 0); // connecte funnel à ourself
  }
  else // complain about arguments
  {
    post ("le message sliders nécessite des arguments");
    post();
  }
}
```

En pseudo-code, notre fonction **sliders ()** exécute les étapes suivantes: Vérifiez si les arguments de la méthode sliders sont valides. Si c'est le cas… S'assurer que le nombre de curseurs demandés est dans une fourchette raisonnable (**0-128**). Supprimez tous les objets créés précédemment par notre objet *js*. Créez les nouveaux objets et connectez-les les uns aux autres. Trouvez notre objet *js* (voir ci-dessous) et connectez-le à notre nouveau *funnel*. Si faux… Affichez un message d'erreur dans la console Max et quittez la fonction.

Notre code JavaScript tire parti de deux caractéristiques importantes de la programmation procédurale, à savoir les instructions conditionnelles (if… else…) et l'itération (boucles for ()). Si vous avez utilisé un autre langage de programmation tel que C ou Java, ces constructions doivent vous être familières. Une référence JavaScript vous aidera avec tous les détails.

Une des premières choses que nous faisons dans notre fonction **sliders ()** est de vérifier quels étaient les arguments du message sliders envoyé par le patcheur. Pour ce faire, nous vérifions la propriété ***arguments*** de la fonction elle-même, par exemple:

```
if (arguments.length) {
// un peu de code ici
}
```

n'exécutera le code entre les accolades que s'il existe un nombre d'arguments différent de zéro dans le message qui a appelé la fonction. Sinon, cette partie du code sera ignorée. De même, vous pouvez accéder aux arguments par numéro sous forme de tableau:

```
a = arguments[0];
```

assignera la variable **a** à la valeur stockée dans le premier argument du message. Dans notre cas, il s'agit du nombre de curseurs que nous voulons créer.

Création et maintenance d'objets

Du point de vue de l'utilisation de *js* pour la création d'objets dans Max, la classe Maxobj nous permet d'utiliser nos variables d'objet pour créer, connecter et détruire des objets. Cela se fait en accédant d'abord à l'objet ***Patcher***, qui est une représentation JavaScript de notre patch Max. L'instruction:

```
this.patcher.remove (thefunnel)
```

indique à *js* de trouver un Maxobj appelé **thefunnel** dans le patcher appelé **this** (qui est toujours le patcher contenant l'objet *js*) et de le supprimer. Le 'this' dans la déclaration est en fait facultatif, mais il convient de noter que vous pouvez utiliser JavaScript pour contrôler des objets dans des patchs autres que celui dans lequel réside l'objet *js*.

Pour créer un objet, nous assignons une variable à un nouveau Maxobj créé par Patcher:

```
thefunnel = this.patcher.newdefault (300, 300, "funnel", a);
```

Dans ce cas, Maxobj **thefunnel** est créé pour être un objet par défaut aux coordonnées **300** par **300** sur la fenêtre du patcheur. La classe de l'objet est fixée sur *funnel*, avec les arguments fixés à ce qui est contenu dans la variable **a**.

Remarque: la méthode **newdefault ()** de l'objet Patcher crée un nouvel objet comme si vous l'aviez créé manuellement à partir de la palette ou du menu contextuel du Patcher. Cela simplifie considérablement la création de scripts. Si vous souhaitez spécifier tous les paramètres de l'objet (largeur d'objet, drapeaux, etc.), vous pouvez utiliser la méthode **newobject ()** à la place.

Les connexions sont réalisées en prenant deux Maxobj et en les liant à l'aide de la méthode **connect ()** à un objet Patcher, par exemple:

this.**patcher**.*connect* (**thesliders** [*5*], *0*, **thefunnel**, *5*)

connectera la sortie la plus à gauche (0) du sixième Maxobj du tableau **thesliders** à la sixième entrée du Maxobject **thefunnel**. N'oubliez pas que la numérotation commence à **0** pour les tableaux et les numéros d'entrée / sortie.

Nous utilisons l'itération et les tableaux pour créer plusieurs objets à la fois, par exemple:

```
pour (k = 0; k <8; k ++)
{
   controlin [k] = this.patcher.newdefault (300+ (k * 100), 50, "ctlin", k + 1);
   thesliders [k] = this.patcher.newdefault (300+ (k * 100), 100, "slider");
   this.patcher.connect (controlin [k], 0, thesliders [k], 0);
   this.patcher.connect (thesliders [k], 0, thefunnel, k);
}
```

générera automatiquement 8 objets *ctlin* et *slider* espacés de 50 pixels sur la fenêtre de patcheur (en partant de la coordonnée horizontale **300**), les connectera les uns aux autres, puis les connectera à l'objet *funnel* référencé par **thefunnel**. Notez que la variable **k** dans notre code JavaScript n'est jamais déclarée, puisque nous l'utilisons uniquement comme variable locale (dans la fonction **sliders ()**) et la réinitialisons à chaque fois que cette fonction est appelée. Dans notre code JavaScript réel dans le patch du didacticiel, le nombre **8** est remplacé par la variable locale **a**, qui représente le nombre de curseurs que nous voulons créer.

Se retrouver dans tout cela

Une chose importante que nous accomplissons dans notre méthode **sliders ()** est la connexion de l'objet *funnel* créé par JavaScript à l'entrée de notre objet *js*. Cependant, notre objet *js* a été créé à la main, et non par notre code JavaScript (ce qui serait impossible, si vous y réfléchissez). Comment lions-nous un Maxobj à un objet créé indépendamment d'un programme JavaScript ?

ourself = this.**box**; // assigner un Maxobj à notre objet js

La propriété 'box' de notre patcheur renvoie un Maxobj se référant à notre objet *js* lui-même! Nous prenons alors la variable ourself et l'assignons à notre objet *js*. Cela nous permet d'établir des connexions avec l'objet contenant notre code JavaScript.

Nous connectons ensuite notre objet *funnel* à notre objet *js* en utilisant notre Maxobj ourself nouvellement assigné:

this.**patcher**.*connect* (**thefunnel**, *0*, nous-**mêmes**, *0*);

Autres méthodes

L'objet *js* de ce didacticiel ne se contente pas de créer et de connecter une surface de contrôle MIDI; il réagit également aux messages de la surface de contrôle ainsi qu'à d'autres messages du patcheur Max. Ouvrez à nouveau le code source de l'objet *js* dans le patch du didacticiel et recherchez la fonction appelée **list ()**:

```
// list - lue à partir de l'objet funnel créé
fonction list (val)
{
   if (arguments.length == 2)
   {
      thevalues [arguments [0]] = arguments [1];
   }
}
```

Comme pour notre fonction **sliders ()**, notre fonction **list ()** vérifie d'abord le nombre de valeurs que nous avons envoyées de Max, par exemple:

```
if (arguments.length == 2) {}
```

L'objet *funnel* établit une liste correspondant au numéro de l'entrée recevant la valeur suivie de la valeur reçue. Par exemple, le nombre **55** arrivant à la deuxième entrée (qui est en réalité l'entrée numéro **1**) déclenchera la liste **1 55** à partir de l'objet *funnel*. Nous vérifions pour nous assurer que nous avons bien deux arguments dans notre message avant de poursuivre dans notre méthode **list ()**, car nous utilisons les deux valeurs de la liste dans la fonction. Nous utilisons le premier argument (quel curseur que nous avons déplacé) pour déterminer quel élément du tableau thevalues nous avons attribuée au second argument (la valeur).

Regardez les fonctions **bang ()** et **reverse ()** dans le code JavaScript.

```
// bang - fait défiler le séquenceur
function bang ()
{
   if (seqcounter> = numsliders) // réinitialise le séquenceur
   {
      seqcounter = 0;
   }
   if (therverse) // lit le tableau à l'envers
   {
      outlet (1, numsliders-seqcounter-1); // envoie notre position dans la séquence
      outlet (0, thevalues [numsliders-seqcounter-1]); // envoie la note en cours
   }
   else // lire à partir du tableau
   {
      outlet (1 sqcounter); // envoie notre emplacement dans la séquence
      outlet (0, thevalues [seqcounter]); // envoie la note en cours
   }
   seqcounter ++; // incrémente la séquence
}
```

```
// reverse - change le sens de la séquence
function reverse (val)
{
   if (arguments.length)
   {
      thereverse = arguments [0]; // retournez
   }
}
```

Notre méthode **bang ()** (qui dans notre patch est déclenchée par un objet *metro*) parcourt une séquence de valeurs d'une manière analogue à l'objet *counter*. Le nombre maximum est défini par le nombre de curseurs que nous avons dans notre patch (défini par **numsliders**). La direction du comptage est toujours ascendante, revenant à **0** lorsque nous dépassons le nombre de curseurs. La fonction **reverse ()** définit une variable (**thereverse**) basée sur les arguments d'un message reverse envoyé par Max. Cela change la façon dont la méthode bang () lit le tableau (**thevalues**) qui stocke les chiffres de notre surface de contrôle des objets *sliders*. Nos deux fonctions **outlet ()** envoient l'index actuel par la sortie droite (**1**) de notre objet *js*, suivie de la valeur de cet index dans la séquence par la sortie gauche (**0**) de notre objet *js*. Notez que nous suivons l'importante convention Max qui consiste à sortir les valeurs des sorties dans un ordre de ***droite à gauche***. Sinon, notre objet *pack* serait déclenché par son entrée gauche avant de recevoir la valeur dont il a besoin dans son entrée droite.

La fonction **outlet ()** affiche la valeur à l'index actuel dans la séquence par la sortie gauche (**0**) de l'objet *js*.

Maintenant que vous savez comment fonctionne le code JavaScript, jouez un peu plus avec le patch. Réfléchissez à la façon dont vous recréeriez le séquenceur en utilisant les objets *table* et *counter* normaux de Max.

Résumé

L'objet *js* vous offre un moyen puissant de créer des patchs Max dynamiquement en JavaScript. La création d'objet est accomplie par l'assignation de variables aux objets ***Maxobj*** créés par un objet ***Patcher***, qui représente le patch en JavaScript. Les méthodes **newdefault ()** et **newobject ()** vous permettent de créer des objets qui peuvent être détruits par une méthode **remove ()**. La méthode **connect ()** vous permet d'établir des connexions de patcheur entre les Maxobjs de notre script. Un Maxobj peut être affecté à l'objet *js* lui-même via la propriété 'box' du patcheur. Lorsque vous concevez des fonctions JavaScript pour qu'elles agissent comme des méthodes pour les messages Max, les arguments transmis avec les messages sont disponibles dans le tableau des arguments de la fonction.