CGI Studio 3.6

Release Date: June 2018

Candera 3.6

To keep backwards compatibility for any interface change in the public API the old interface has been kept and marked as deprecated. The deprecated interfaces will be removed in future versions. Compiler warnings will be generated when using a deprecated interface through the usage of Candera Macro CANDERA_DEPRECATED_3_6_0.

Candera/FeatStd CMake Build System Changes

Added flags:

Name	Description	Possible values
CANDERA_TEXTENGINE_NATIVE_LINE HEIGHT_ENABLED	Enables the native line height metrics of the font engine. If disabled then 120% of EM size is applied as line height which is compatible with Adobe.	ON/OFF

Candera Engine 2D

Node2d GetComputedBoundingRectangle, IsInsideBoundingRectangle, IsPickIntersectingBoundingRectangle and GetEffectiveBoundingRectangle interface extended

The method 'virtual void Node2d::GetComputedBoundingRectangle(Rectangle& boundingRectangle, bool ignoreInvisible = false) const' has been extended by the additional parameter ignoreInvisible. The default value of this parameter is false for full usage backwards compatibility. If the parameter is set to true only visible parts of the scene will be considered. This also means that clipped areas are not considered for picking and/or bounding box calculations. As a consequence it is now possible to pick only visible parts of the scene.

Pivot point for Transformable2D

New methods to Transformable2D have been added that allow to manipulate the real pivot point.

Those are Transformable2D::SetPivotPoint and Transformable2D::GetPivotPoint .

Candera Engine 3D

Generic Transform is Optional

The Property "Generic Transform" of a 3D Node became optional. The reason for that is to have as few matrix multipilcations as possible which saves performance. Further wise Generic Transforms are only used for special needs.

Pivot point for Transformable

New methods to Transformable have been added that allow to manipulate the real pivot point.

Those are Transformable::SetPivotPoint and Transformable::GetPivotPoint .

Transitions

Sequential Transitions - Extended Hint Constructor

The constructor for Transition Hints has been extended in order to reflect the newly added ability to execute Transitions in a sequential manner.

The added parameters are:

```
const FeatStd::Optional<FragmentStrategy>& activationStrategy
const FeatStd::Optional<FragmentStrategy>& deactivationStrategy
const FeatStd::Optional<Float>& activationDelay
const FeatStd::Optional<Float>& deactivationDelay
```

FragmentStrategy is a new enum type consisting of:

- Normal
- Early
- Late

Transition Rules - Check if Rule is Bidirectional

The Rule class has a new method called bool Rule::IsBidirectional() const, which returns, whether the Transition Rule is set to be bidirectional or not.

Textengine

AssetFontStore support for Freetype streams

AssetFontStore is extended to support two font load strategies:

- AssetFontStore::LoadStrategySelector::PreloadStrategy (the old behavior before the extension): entire fonts are loaded into memory once and font data is accessed from there by Text Engine.
- AssetFontStore::LoadStrategySelector::OnRequestLoadStrategy (new behavior, selectable through a strategy selector): small font portions are streamed directly from the asset repository when Freetype (or indirectly Harfbuzz) needs them.

Use DefaultLoadStrategySelector::SetDefaultFontLoadStrategy to select one of these load strategies. Use DefaultLoadStrategySelector::SetDefaultFontLoadStrategyExceptionList to exclude a list of fonts from the selected font load strategy.

Example:

```
    // Load fonts with OnRequestLoadStrategy
    selector.SetDefaultFontLoadStrategy(
Candera::AssetFontStore::LoadStrategySelector::OnRequestLoadStrategy);
    // "Bitstream Vera Sans" font shall be loaded with PreloadStrategy:
    const Char* g_fontName[2] = { "ConstructionKit##ConstructionKit#Resources#Fonts#Bitstream Ve
    selector.SetDefaultFontLoadStrategyExceptionList(g_fontName);
    fontStore.SetLoadStrategySelector(&selector);
```

Courier applications can use Courier::ViewHandler::SetFontStoreProviderCallback to provide specific settings for the Candera::AssetFontStore in use.

Support of Monotype's WorldType-Shaper

Candera supports the WorldType-Shaper of Monotype now. The integration is done to support the font engine 'iType'. With this release complex texts can be shaped. The behavior is similar to HarfBuzz with the difference of a huge performance boost. WT-Shaper may have a higher memory consumtion as a trade-off to performance. The implementation concept is aligned with HarfBuzz to support a flexible exchangeability.

With this the default shaper of iType switches from ComplexScript to WT-Shaper. The corresponding CMake flag CANDERA_TEXTSHAPER has been extended by the entry WorldType-Shaper.

Courier V3.6

Data Context

Support for a Candera node 2D/3D based local data context for binding sources. Behavior/widget property bindings will be bound to the nearest binding source instead of the global binding source from the model. This enables:

- The usage of courier data binding within different contexts (e.g. each item of a list can act as a binding source for controls).
- Using standard Courier data binding for data items within list items.

Data Context Item

Each binding source can be marked as a Data Context Item by setting the flag dataContextItem to true. As a result the binding source data struct will be derived from the Data Context Item base class. Data Context Items can be handled with shared pointers. However, they will be cloned automatically when they are used as a list item. this ensures that there is no need to add synchronization handling in the view. The cloning is only performed on the struct. Each member will be cloned by using its copy constructor. If this results in a shallow copy (e.g. like for other shared pointers, reference, pointers, ...) then the instance is shared between the model and the view. Synchronization handling has to be considered within the application code for those items. The Data Context Item also enables the binding to generic lists with items of different dynamic data type (e.g. for inhomogeneous lists).

Courier List model improvements

The list model has been improved:

- Dynamic list fragment: If no maximum and no window size is given in the xml specification then the list fragment will use a dynamic fragment size with variable maximum length.
- No need to provide a maximum size for list fragment: If a window size is given in the xml specification then no maximum size has to be provided (which also had been ignored in the past in that case).
- Modified flags for list items: In addition to the single modified event approach the list model interface (including list fragments) provide a set of modified flags to enable a single update message whenever several list items are modified (e.g content or position). All flags will be reset after the update message of the binding source is sent.

- Automatic setting of list item modified flag: Whenever a list item is modified with the list model interface its modified flag is set in the modified flag set.
- Manual modified flag handling of list items: To handle false positives of modified flags an interface is available to reset the modified flag manually.
- Modification of data item within a data context: To enable the notification of the model of data item change request within a local data context binding source (e.g. a single data item within a list item has to be changes to a specific value) the UpdateModelMsg has been extended by the ChangeDataItem request. This allows the modification of list content with the highest possible precision simply with standard Courier data binding.

FeatStd 3.6

FeatStd Util

Variant explicit constructor

The Variant constructor have been changed from conversion constructor to explicit constructor. Assignment similar to the following will no longer compile:

```
FeatStd::Variant variant = 1;
```

Furthermore, the direct usage of values (e.g. Int values) as paremeters will no longer compile:

```
FeatStd::Optional<FeatStd::Variant> optionalVariant = FeatStd::Optional<FeatStd::Variant>(1)
```

Since the constructor is no longer inserted automatically by the compiler it has to be inserted manually to make

```
FeatStd::Variant variant = FeatStd::Variant(1);
FeatStd::Optional<FeatStd::Variant> optionalVariant = FeatStd::Optional<FeatStd::Variant>(
FeatStd::Variant(1));
```

the code compile again:

Variant explicit constructor

The list of supported types that a FeatStd::Variant can handle has been extended. Beside the converting getter methods also TryGet methods are provided that will indicate if the, conversion was limited by the value range of the result type (e.g. a value 256 converted to a UInt8 will result in the value 255 due to the range limit of UInt8).

SceneComposer 3.6

New Feature

Description of New Feature