There is no Spoon

Machineri Nirvana

A Guide through the Galaxy by Marvin Siddharta

Index

Find the name of the chapter you'd wish to read up on here, before diving into the 'big pie' ...

7. ListModule

Level4: Implements most of the logic for our DBAdmin module and basically every single Grid we have in Magix. This class is the foundation for probably most Magix screens you've ever seen, since every Grid within the system, virtually, is built using this class. Have so many bells and whistles, it could probably need its own book, not to mention some serious refactoring. Till at least the refactoring parts are over, I think I'll resist the temptation of going 'over the board' in documenting this, since first of all everything will change later, and you shouldn't fiddle too much directly with this bugger yourself. There's also PLENTY code samples around ANYWHERE in magix of usage of this class, so have fun :)

8. DBAdmin_Controller

Level2: Contains the logic for the DBAdmin module(s), which is also the Grid/CRUD-Foundation system in Magix. Contains many useful methods and ActiveEvents for displaying either Grids or editing Single Instances of Objects. Can react 100% transparently on ActiveTypes. Has support for Meta Types, meaning types where you're more in 'control', but must do more of the 'arm wrestling directly'

9. ColumnTypesPlugins_Controller

Level3: Contains some template columns for the Grid system for you to use in your own Grids

10. Documentation_Controller

Level2: Contains the logic for our 'Class Browser' which can browse all the classes in the system and make some changes to them by enabling them and disabling them by either overriding specific events or by disabling entire controllers or modules all together

11. Email_Controller

Level2: Implements logic for sending email using SMTP through the .Net classes

12. FileExplorer_Controller

Level2: Contains Logic for file explorer. The file explorer is a component where you can browse the file folder system on your web server, remotely, and do many operations. Such as editing CSS files, uploading images and such. PS! To use this Module it is IMPERATIVE that you've given the 'NETWORK SERVICE' account 'Full Access' to at the very least the 'media/' folder, or whatever folder you plan to use the explorer on on your web server

13. Logger_Controller

Level2: Contains logic for Logging things

14. MetaAction_Controller

Level2: Contains logic for Actions which are wrappers around ActiveEvents for the end user to be able to raise his own events, with his own data in the Node structure. The MetaAction system is at the core of the Meta Application system wince without it the end user cannot create his own types of events or Actions

15. MetaObject_Controller

Level2: Contains logic for editing, maintaining and viewing MetaObjects. MetaObjects are at the heart of the Meta Application System since they serve as the 'storage' for everything a view updates or creates through interaction with the end user

16. CommonActions_Controller

Level2: Contains common end user useful actions which doesn't really belong any particular place, but which can still be immensily useful for 'scripting purposes'. Perceive these as 'plugins' ... ;) - [Or extra candy if you wish]. Often they're 'simplifications' of other more 'hard core' Active Events

17. CreateDefaultInitialActions_Controller

Level2: Contains Application Startup code to create the defaul Actions unless they're already there

18. MetaView_Controller

Level2: Contains helper logic for viewing and maintaing MetaViews, and related subjects. MetaViews are the foundation for the whole viewing parts of the Meta Application system. A MetaView is imperativ for both being able to collect new data and also for viewing existing data. The MetaView defines which parts of the object you can see at any time too, which means you can use it to filter access according to which Grid the user is having access to, for instance. This controller contains logic for editing and maintaining MetaViews, plus also direct usage of MetaViews

19. PDF_Controller

Level2: Contains logic for creating PDF files and dowloading to Client or saving locally

20. Logging_Controller

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

21. LoginOut_Controller

Level2: Login and Logout Controller

22. FileSystem_Controller

Level2: Tiny helper for menu item in Administrator Dashboard to view the File System Browser

23. InitialLoading_Controller

Level2: Class for taking care of the Magix.Core.InitialLoading message, meaning the initialization of the page on a per User level initially as they're coming to a new 'page' or URL ... Basically just handles Magix.Core.InitialLoading and mostly delegates from that point ...

24. SliderMenu_Controller

Level2: Helps feed the SliderMenu in Front-Web to get its Items to build its structure upon

25. TipOfToday_Controller

Level2: Creats our default Tooltips or Tutorial rather

26. OpenID_Controller

Level2: Controller implementing the OpenID logic for both Relying Party and OpenID Provider logic. Every User in Magix can use the website where Magix is installed as an OpenID provider by typing in the root URL of the web application and append ?openID=username In addition you can associate as many OpenID tokens as you wish to your User and then use these other OpenID providers to log into Magix.

27. PageLoader_Controller

Level2: Controller responsible for loading up 'Page Objects', and doing initialization around pages and such as they're being loaded

28. WebPart_Controller

Level2: Initializes WebParts and such while being injected onto WebPage

29. AdministratorDashboard_Controller

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

30. ChildExcerpt_Controller

Level2: Helps out with some of the editing and using of the ChildExcerpt Module Type

31. EditUser_Controller

Level2: Here mostly for Editing User, and also to some extent creating default users and roles upon startup if none exists

32. AdminDashboardHeader_Controller

Level2: We'll 'lock' the Header control since it can be VERY annoying sometimes due to the DB Manager, which seriously needs to be refactored here BTW ...

33. EditRoles_Controller

Level2: Here mostly to serve up grids and such for editing of Roles in the system

34. EditTemplates_Controller

Level2: Controller for editing Templates. Contains all the relevant event handlers and logic for editing your templates

35. Dashboard_Controller

Level2: Main 'router' in dispatching important Dashboard functionality

36. EditPages_Controller

Level2: Contains logic for Editing WebPage objects and such for administrator

37. Access_Controller

Level3: Helps out sorting out which Pages and Menu Items which Users and Roles and such have access to

38. Settings_Controller

Level2: Helper logic to assist with Settings and updating of settings. PS! Unless you have this controller in your Application Pool, Enabled, then direct changes, through the DB Admin module to Settings will NOT affect the Application Pool before it's being re started ...

39. Signature_Controller

Level2: Signature Plugin Control for Publishing system to allow for loading of a different plugin module from e.g. the click of a row button or a single-view button etc.

40. TalkBack_Controller

Level2: Contains helper logic for the TalkBack module [forums] in Magix

41. ToolTip_Controller

Level2: Contains helper logic for displaying and maintaining previous/next Tip of today and such

42. Separator

Level2: Encapsulates a [dead] html Horizontal Ruler [hr] element for you if you need one

43. TipOfToday

Level2: Shows one tip of today with the option for the User to browse forward or backward to read more. Raises 'Magix.Core.GetPreviousToolTip' and 'Magix.Core.GetNextToolTip' to get its next and previous tips

44. Tree

Level2: Shows a Tree module for the end user for him to navigate and select single nodes from. Change its properties by passing in 'TreeCssClass' or 'NoClose'. 'Items' should contain the tree items in a hierarchical fashion with e.g. 'Item/i-1/' containing 'Name' 'CssClass' and 'ToolTip'. 'Name' being minimum. Child items of 'Items/i-1' should be stored in the 'Items/i-1/Items' node. Will raise 'GetItemsEvent' upon needing to refresh for some reasons, and 'ItemSelectedEvent' with 'SelectedItemID' parameter as selected item ID upon user selecting an item

45. MessageBox

Level2: Implements logic for showing the end user a message box, asking for confirmation or something similar when needed. 'Text' will be the text shown to the user, which he should read and take a stand in regards to. Dropping the 'Cancel' Node sets Cancel to in-visible if you wish. 'OK/Event' will be raise with 'OK' node if OK button is clicked. 'Cancel/Event' will be raised with 'Cancel' node if Cancel button is clicked

46. Clickable

Level2: Encapsulates a Clickable Component, basically a button. Which you can load as a module, and trap clicks to. Set its 'Text', 'ButtonCssClass', 'Enabled' and 'ToolTip' params to customize it till your needs. Will raise 'Event' when clicked

47. Header

Level2: Encapsulates a header [h1] control for you to create headers for your pages and apps. Load it and raise 'Magix.Core.SetFormCaption' to set the header

48. ImageModule

Level2: Control for displaying images in your app. Either clickable or static images. Pass in 'ImageURL', 'AlternateText', 'ChildCssClass' and 'Description' to modify it according to your needs. Description, if given, will add a label underneath the image. Use 'Seed' to have multiple Images on same page and to separate between different instances of them. If 'Events/Click' is given, it'll raise that event upon clicking the image

49. ViewClassContents

Level2: Will show all objects of type 'FullTypeName' in a Grid from which the user can filter, edit, change values, delete objects, create new objects and such from. Basically the 'show single table' logic of the Magix' 'Database Enterprise Manager'. If 'IsCreate', will allow for creation of objects of the 'FullTypeName' by the click of a button

50. ViewListOfObjects

Level2: Basically the same as ViewClassContents, though will only show objects 'belonging to a specific object [ParentID] through a specific property ['PropertyName'] and allow for appending, and not creation of new objects of 'FullTypeName'. Raises 'DBAdmin.Data.GetListFromObject' to get objects to display in Grid. Override the Append button's text property with 'AppendText'. Other properties are as normal from mostly every grid in Magix such as 'IsDelete', 'IsRemove' etc

51. ViewSingleObject

Level2: Contains the logic for editing and viewing one single ActiveType object, with all of its properties. Can become initiated in two different states, one of which is 'edit object reference from another object' which will allow for changing and removing the reference, the other is plain old 'edit the thing' mode. Supports 'ChildCssClass' and several other properties. Most of the common properties from the Database Enterprise Manager is included. Will raise 'DBAdmin.Form.ChangeObject' if user attempts to change the reference. Will raise 'DBAdmin.Data.RemoveObject' when object reference is removed. Changing the reference or removing it is only enabled if 'IsChange' and/or 'IsRemove' is given as true

52. FindObject

Level2: Basically identical to ViewClassContents, with one addition. This module will show a 'filter field' to the end user which the user can use to 'search for items' from within.

53. BrowseClasses

Level2: Contains UI for letting the end user browse the ActiveType classes within his system as a Database Enterprise Management tool. Allows for seeings all classes in a Tree hierarchy and letting him select classes, which again will trigger editing of records in that class. Kind of like the Database Anterprise Management tool for Magix. Will raise 'DBAdmin.Data.GetClassHierarchy', which you can handle if you have 'meta types' you wish to display

54. ConfigureColumns

Level2: Allows for editing of 'visible columns' per type level. Will show a form from which the end user can tag off and on the different columns he wish to see in his grid. Databinds towards 'WhiteListColumns' and raises 'DBAdmin.Data.ChangeVisibilityOfColumn' when visibility of column changes with 'Visible', 'FullTypeName' and 'ColumnName' as parameters

55. ConfigureFilters

Level2: Will allow the end user to configure his filters, meaning what filter to filter each column by. There are many different types of filter in Magix. LIKE, Equal and so on. This form allows you to globally set the filters for specific types on specific columns of those types

56. ShowClassDetails

Level2: Doxygen helper class for displaying documentation about members of classes and such for our documentation system. Takes in 'FullName', 'Description' and so on. Will create a grid if displaying items according to the structure given

57. RichEdit

Level2: Contains the UI for the RichEditor or WYSIWYG editor of Magix. That little guy resembling 'word'. Specify a 'SaveEvent' to trap when 'Text' is being edited.

58. Explorer

Level2: Containe the UI for the Explorer component, which allows you to browse your File System on your server, through your browser. Basically a File System Explorer kind of control, which allows for renaming, deleting, and editing [to some extent] the files in your installation. Can be instantiated in Select mode by setting its 'IsSelect' input parameter. If 'CanCreateNewCssFile' is true, the end user is allowed to create a new default CSS file which he can later edit. 'RootAccessFolder' is the root of the system from where the current user is allowed to browse, while 'Folder' is his current folder. The control does some basic paging and such, and has support for will raise 'Magix.FileExplorer.GetFilesFromFolder' to get its items. The module will raise the value of the 'SelectEvent' paeameter when an item has been selected. The module supports browsing hierarchical folder structures

59. EditAsciiFile

Level2: Kind of like Magix' version of 'Notepad'. Allows for editing of textfiles or text fragments, and allow for saving them

60. Slider

Level2: Contains the UI for our SlidingMenu module, used in the administrator dashboard. Takes a recursive 'Items' structure containing 'Caption' and 'Event' which will be raised when clicked ['Event']

61. MetaView_Single

Level2: UI parts for showing a MetaView in 'SingleView Mode'. Basically shows a form, with items dependent upon the look of the view. This is a Publisher Plugin module. This form expects to be given a 'MetaViewName', which will serve as the foundation for raising the 'Magix.MetaView.GetViewData' event, whos default implementation will populate the node structure according to the views content in a Key/Value pair kind of relationship. This will serv as the foundation for the module to know which types of controls it needs to load up [TextBoxes, Buttons etc]Handles the 'Magix.MetaView.SerializeSingleViewForm' event, which is the foundation for creating new objects upon clicking Save buttons etc.This is the PublisherPlugin you'd use if you'd like to have the end user being able to create a new MetaObject

62. MetaView_Multiple

Level2: UI parts for showing a MetaView in 'MultiView Mode'. Basically shows a grid, with items dependent upon the look of the view. This is a Publisher Plugin module. Basically a completely 'empty' module whos only purpose is to raise the 'Magix.MetaType.ShowMetaViewMultipleInCurrentContainer' event, whos default implementation will simply in its entirety replace this Module, hence not really much to see here.This is the PublisherPlugin you'd use if you'd like to see a 'list of MetaObjects' on the page

63. LogInOutUser

Level2: PublisherPlugin containing most of the UI for allowing a user to login/out of your website. Can be set into both OpenID mode and 'only native mode' or both. Raises 'Magix.Publishing.GetStateForLoginControl' to determine the state of the module, meaning if it should show both OpenID logic and native logic or only one of them. Will raise 'Magix.Core.UserLoggedOut' if user logs out and 'Magix.Core.LogInUser' if the user tries to log in. The 'Magix.Core.LogInUser' default implementation again will raise 'Magix.Core.UserLoggedIn' if it succeeds. It'll pass in 'OpenID' if user has chosen to log in with OpenID and 'Username'/'Password' if user choses to login natively

64. Header

Level2: A 'header' PublisherPlugin. Basically just an HTML h1 element

65. TopMenu

Level2: PublisherPlugin containing a conventional 'File Menu' type of menu which normally is expected to find at the top of a page, which will 'drop down' child selection boxes for being able to select children. Useful for conventional applications, which should look like legacy code, or something. Takes the exact same input parameters as the SliderMenu PublisherPlugin

66. SliderMenu

Level2: PublisherPlugin containing the UI for the Sliding Menu Publisher Plugin. Meaning the default menu to the left in the front-web parts, which you can choose to inject into a WebPartTemplate in one of your Templates if you wish. Basically just loads the items once through raising the 'Magix.Publishing.GetSliderMenuItems' event, which should return the items as a 'Items' list, containing 'Caption', 'Event', 'Event' [Event name] and 'Event/WebPageURL' which normally will contain the page's URL

67. Content

Level2: A 'content' PublisherPlugin. Basically just a text fragment, that'll be edited through the Magix' RichText or WYSIWYG Editor

68. ChildExcerpt

Level2: Will show an excerpt of all its children, sorted with newest first, showing a maximum of PagesCount items. Kind of like a front page to a blog or a news website or something

69. EditSpecificTemplate

Level2: Allows for editing of WebPageTemplate objects. Contains most of the UI which you're probably daily using while adding and creating new templates and such

70. EditSpecificPage

Level2: Allows for editing of one single specific WebPage in the system. Contains most of the UI which you're probably daily using while adding and creating new pages and such

71. Sign

Level2: A Signature module, basically a place where the end user can 'sign his name' to confirm a transaction of some sort. Will load up a big white thing, which can be 'drawn upon', together with two buttons, OK and Cancel which will raise the 'CancelEvent' and the 'OKEvent. 'OKEvent' will pass in 'Signature' being the coords for all the splines that comprises the Signature, which can be stored and later used as input to this Module, which will again load those splines in 'read-only mode'

72. Forum

Level2: Basically a 'Forum module' which allows for posting and reading and replying to other people's opinions about 'whatever'. Not very good on its own, please use through TalkBack Controller to save head aches

73. Login

Shows a login box with username/password/openid combo. Username/password has preference, but if only OpenID is given, the system will attempt at login you in using the OpenID claim. Note, this is not the Publisher Plugin login box. This is the 'main system one', which you get by going to ?dashboard=true

74. Settings

Level2: Wrapper class for common/global settings within a Magix-Brix application. Use the overloaded this operator to access Settings through code

75. TipOfToday

Level3: Helper class for Tip [Today's Tips] logic

76. LogItem

Class encapsulating log items in Magix. Magix has automated logging components which will log certain aspects about your system and its behavior. These log items are stored in your database through using this active type

77. MetaObject

The storage for the 'Meta Application System' in Magix. Every time you create a new object, using a Meta View or something similar, a MetaObject is being created to hold this data. A Meta Object is basically an object 'without structure', where you can, by editing your views, maintain the structure you wish for your data. In such a way you can create your own Meta Applications, using Magix, without being imposed onto an existing data-structure in any ways

78. Action

Contains the serialized content of the Meta Action system in Magix. One Action is a wrapper around an Active Event, as described in the O2 Architecture Document. Basically, an Action can be thought of as a 'serialized version' of an ActiveEvent being raised, which you then can choose to raise whenever you wish, at the trigger of a button for instance. By creating your own actions, you can effectively implement your own type of logic and flow in Magix without having to code. Every Action must have a unique name, and is often either referenced by its Name or its ID property.

79. MetaView

Contains the logic for our views in our 'Meta Application System'. A MetaView is something that defines what the user is supposed to see in regards to MetaObjects. Basically the UI 'representation' of your object graphs and properties. A MetaView can become the 'engine' for a WebPart through the MetaView_Single and MetaView_Multiple plugins for Magix

80. WebPageTemplate

Serves as a 'recipe' for a WebPage. Every WebPage is built from one of these. Contains the definition of which module to instantiate for instance, but not which parameters to send into it. Also contains the positioning of the WebParts and other common features, such as CSS classes and such. A Template is just that, a 'template' for your WebPages

81. WebPart

Is one instance of a container on a WebPage. Baically one 'module' if you wish. A WebPage is created of several WebParts, each WebPart again is built from a corresponding WebPartTemplate associated with the WebPageTemplate the page itself is built upon. This class encapsulates the WebPart of this relationship, and hence serves as a wrapper for WebPart settings basically on a per page level

82. WebPartTemplate

Serves as a 'recipe' for WebParts. Contains stuff such as positioning of webpart, and name of module to inject. Every WebPageTemplate has a list of these guys, which serves as recipe for how the WebPage's WebParts should be built, and which Plugins to load up

83. WebPageRoleAccess

Gives access to a page according to a specific role. This is the core class in Magix in regards to authorizing access to specific pages. Access in Magix is associated with a User, his Role and the WebPage the user tries to access. You cannot authorize on a more fine-grained level out than WebPages, out of the box. This class encapsulates the authorization logic in Magix. By default everyone have access to everything. If one Role is explicitly granted access, then all other roles will be denied access, unless they too are granted access. In addition access is 'inherited' from the 'Mother Page'. Meaning a child page will be default have the same access rights as its Mother page, unless the child page itself starts granting and denying access to specific pages

84. OpenIDToken

Encapsulates an OpenID Token. Meaning an OpenID Claim. Magix has support for serving as both an OpenID Relying Party and an OpenID Provider. This class serializes the OpenID Claims for users when using Magix as an OpenID Relying Party, and associating these claims with specific users. This means you can associate an OpenID claim to your user and then use the OpenID Token to log into Magix later

85. User

Encapsulates one user in Magix' Publishing system. Inherited to add more, specific to the publishing system, to the class

86. WebPage

Represents one web page in our system, or one unique URL if you wish. Contains a list of WebParts which again are areas of the screen within your web page. Every page must have at least one webpart, but can have many more. This class encapsulates the logic of 'Pages' in Magix, which is probably easy to understand within the context of publishing or CMS systems. Anyway, a page in Magix might also be a container for your application, plugins and WebParts. WebParts and Plugins again can be either your own creations through MetaViews or something similar, or actual C# code written plugins for your system. One page is often easy to understand if you can perceive it as 'one URL', while it is still much more powerful than any 'CMS Pages' or 'Publishing Pages' out there.Every page is built upon a 'recipe' which is the WebPageTemplate class. The WebPageTemplate class contains the logic for which plugin type it should ue etc, while the WebPage contains the settings for the type of plugins instantiated upon opening it

87. Posting

Level2: One Talkback posting. If Parent is null, this is a top-level posting, and the Children collection might have content. If not, it's a child itself of another top level posting, as in a 'reply'

88. UserSettings

Level2: Settings stored on a 'per user level'. Useful for storing simple fact on a per user level

89. UserBase

Level2: A User is a single person using your website, being registered with a username and a password, and hence no longer 'anonymous'. A user is normally recognized through his 'Username', and he has a password which he can use to log into the system, normally. This class encapsulates that logic. PS! This class supports inheriting [though it's not really entirely stable quite yet ...!]

90. Role

Level2: Contains the roles i the system. Every user belongs to a 'role' which gives him rights in regards to some aspect of functionality or something. All autorization, by default in Magix, will go through this Role class, and which specific roles the currently logged in user belongs to

91. Criteria

Level3: Abstract base class for all data storage retrieval criterias. Also contains several handy static constructors for easy creation of data storage retrieval criterias.

92. Transaction

Level3: Implements transactional support for your updates and inserts. Use through the C# using keyword to get automatic rollbacks. Or implement finally yourself in your code. Remember to call 'Commit' before Transaction is lost, since otherwise. Caputt. Default is Rollback

93. ActiveTypeAttribute

Level3: Mark your well known types or entity types [or more correctly said; Active Types ;)] with this attribute to make them serializable. In addition you must inherit from ActiveType with the type of the type you're creating as the generic type argument. Notice that this attribute is for classes, you still need to mark every property that you wish to serialize with the ActiveFieldAttribute.

94. ActiveType-g

Level3: Inherit your well known types or entity types - the types you want to serialize to your database from this class giving the generic argument type as the type you're creating. Notice that you also need to mark your types with the ActiveRecordAttribute attribute in addition to marking all your serializable properties with the ActiveFieldAttribute.

95. ActiveFieldAttribute

Level3: Used to mark entity objects as serializable. If a property is marked with this attribute then it will be possible to serialise that property. Notice that you still need to mark you classes with the ActiveRecordAttribute. Also only properties, and not fields and such can be marked as serializable with this attribute.

96. Adapter

Level4: Abstract base class for all Database Adapters in Magix-Brix. If you wish to build your own data adapter then inherit from this class and implement the abstract methods, add up a reference to the dll and change the data-configuration line in your configuration file and it should work. Class provides some common functions needed for all database adapters like caching, instantiation and such in addition.

97. TransactionalObject

Baseclass for 'internal usage' behind ActiveTypes. Definitely candidate for refactoring. In general do NOT reference any members from here directly, except the ID of course

98. MSTransaction

Level4: Implementation of Transaction class for MS SQL DataAdapter

99. MSSQL

Level4: Microsoft SQL Server Database Adapter, which probably works with 2005 and any later, for Magix-Brix. Contains all MS SQL specific logic needed to use Magix-Brix together with MS SQL. This is the default DataAdapter in use in Magix

100. Helpers

Level3: Static helper class for data-storage Adapter developers. If yo're fiddling around here, you'd better know what you're doing ...!! ;)

101. StdSQLDataAdapter

Level4: Common logic for all Database adapters that relies on standard SQL syntax. Wrappers for creating SQL text for derived data adapters that uses RDBS that relies on standard SQL syntax. Sorry, I have to prioritize what I comment. Needs refactoring though, that's one thing ..... but so does the entire Magix.Data namespace ... :(

102. ActiveModule

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your Modules, such as RaisingEvents etc. Inherit your ActiveModules from this class to simplify their usage

103. ActiveEvents

Level3: Class contains methods for raising events and other helpers, like for instance helpers to load controls and such. Though often you'll not use this directly, but rather use it through helper methods on your ActiveControllers and ActiveModules

104. ActiveModuleAttribute

Level3: Mark your Active Modules with this attribute. If you mark your Modules with this attribute you can load them using the PluginLoader.LoadControl method. This is the main attribute for being able to create ActiveModules

105. ActiveControllerAttribute

Level3: Mark your controllers with this Attribute. Notice that an Active Controller must have a default constructor taking zero parameters. This constructor should also ideally execute FAST since all controllers in your Magix-Brix project will be instantiated once every request.

106. ActiveController

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your controllers, such as Loading Modules, raising events etc. Inherit your controllers from this class if you'd like to add more 'power' to them

107. ActiveEventAttribute

Level3: Mark your methods with this attribute to make then handle Magix.Brix Active Events. The Name property is the second argument to the RaiseEvent, or the "name" of the event being raised. You can mark your methods with multiple instances of this attribute to catch multiple events in the same event handler. However, as a general rule of thumb it's often better to have one method handling one event

108. ActiveEventArgs

Level3: EventArgs class that will be passed into your Magix-Brix events - the methods you mark with the ActiveEvent Attribute. The Extra property will contain the "initializationObject" passed into the RaiseEvent.

109. PluginLoader

Level4: Helps load UserControls embedded in resources. Relies on that Magix.Brix.Loader.AssemblyResourceProvider is registered as a Virtual Path Provider in e.g. your Global.asax file. Use the Instance method to access the singleton object, then use the LoadControl to load UserControls embedded as resources. Kind of like the Magix' version of Page.LoadControl. Can be used directly by you, if you really know what you're doing though. In general, I'd say DON'T ...!!

110. AssemblyResourceProvider

Level4: Helper class to make it possible to load controls (and more importantly) UserControls which are embedded as resources in DLLs. Not intended for direct usage, but will be in 'the background' and making sure you can load ActiveModules as resources from your DLLs

111. PublisherPluginAttribute

Level3: I'd be highly surprised if this is not your first entry to Magix in C#. This is the PublisherPlugin attribute, which you can use to create your own plugins for the Publishing system within. Implement this attribute on your ActiveModules and VOILA! They'll surface up as selections in your WebPageTemplate editing operations and be usable as plugins in your system. Probably the easiest way on the planet to create a plugin for any kind of system out there. Especially in combination with the logic behind the ModuleSettingAttribute

112. ModuleSettingAttribute

Level3: Wraps a setting property for a PublisherPlugin

113. PeriodCollection

Level3: A collection class of Period types. Contains algebraic methods for OR, AND, XOR, NOT. Makes algebraic operations on collection of Period objects very easy and intuitive. Can with for instance one line of code OR two collections together to find the logically OR'ed result of these two different collections. Very useful for manipulating dates and such

114. LazyList-g

Level3: Helper class for Lazy Loading of Child ActiveTypes objects. Basically just a list generic type, that'll not load objects before needed. Useful for using as properties for list of child objects in your ActiveTypes

115. Node

Level3: Helper class to pass around data in a "JSON kind of way" without having to convert to JSON strings. Create a new instance, and just start appending items to it like this; Node n = new Node();
n["Customer"]["Name"] = "John Doe";
n["Customer"]["Adr"] = "NY";
This is at the core of Magix, being the 'protocol' we're using to pass data around within the system. If you don't understand this class, you're in trouble! Make sure you understand, at least roughly, what this class does if you'd like to code C# for Magix

116. SingleContainer

Level2: Contains the logic for the main Viewport in Magix. A viewport can be seen as your 'design' and contains all the different logic for being able to load and unload modules and such

117. AspectModal

Aspect useful for making 'modal' controls. A Modal control will obscur the contents behind it by creating a div that is semi-transparent (most often) and fill the entire viewport (browser area). This effect makes it impossible to click controls behind it on the page.

118. AspectAjaxWait

Ajax Wait Aspect. Use this one whenever you've got an Ajax Control that you know will spend a lot of time on the server-side executing its logic. You can either have the client-side logic create its own DOM element to serve as a 'blackout' when it's supposed to display the Ajax Wait DOM element, or you can attach an existing in-visible DOM element to it by using the Element property.

119. AspectBase

Base class for all Aspects in Magix UX. Implements common functionality.

120. AspectDraggable

Aspect for creating controls that can be moved around on the screen using your mouse. It is also possible to handle the Dragged event and track when the dropping of the control occurs.

121. SlidingMenu

This widget is a 'menu looka-like kinda control'. Its purpose is to be an alternative way of displaying hierarchical choices for your users. But instead of 'popping up' like menu items does in a conventional menu, they will 'slide in' from the side, making a much more efficient way of handling your space on your web pages. For most navigational purposes, this widget is a far better bet than the conventional Menu widget.

122. SelectList

A DropDown list type of control, although it can also be set into a non-drop down mode. Basically multiple choices type of widget. To some extent, it overlaps the logical functionality of the RadioButton widget, although the SelectList is more useful for cases where you have a massive number of choices, like for instance choose one out of 300 different languages, while the RadioButton is more useful for cases where you have fewer choices, such as choose 'coffee, tea or water'. Add up your choices by adding up ListItems inside of your SelectList.

123. SlidingMenuLevel

Child control of a SlidingMenuItem. Will contain the SlidingMenuItems of a SlidingMenuItem. If you have items which have children, then you must create an item of type SlidingMenuItem and add those children inside of that object. This is true both for .ASPX markup and SlidingMenu hierarchies created in code.

124. SlidingMenuItem

A single sliding menu item. One instance. A SlidingMenu are basically composed out of a whole hierarchy of these types of widgets.

125. MultiPanelView

Instances of items within a MultiPanel. These are the items a MultiPanel is made of. Most scenarios you would use this control within would have only one of the MultiPanelViews visible at the same time.

126. MultiPanel

A MultiPanel is a collection of MultiPanelViews, often used to create TabControls or similar constructs. Often combined with TabStrip and TabButton. This control can be configured extensively to mimick a lot of different scenarios. Hence we don't call it TabControl, although that happens to be one of the sub-scenarios you can create with it.

127. RadioButton

A two-state widget type that can be grouped together with other widgets to mimick multiple selections. Like for instance; "Would you like to have coffee, tea or water" is a perfect example of where you would want to use RadioButtons. If you want to group specific radio buttons together, you must give them the same GroupName property. If you do, then only one of these RadioButtons can at any time be 'selected'.

128. Panel

A container widget for displaying other widgets inside of it. Will render as a div by default, but the specific tag this widget will render, can easily be overridden by changing the Tag property. You can choose to render your panels as paragraphs (p...) for instance. If you only need to display text on your page, and you need to use WebControls for this, you should use the Label control and not the Panel control.

129. TreeItem

One single item within a TreeView widget. A TreeView is basically nothing but a collection of these types of controls. If you override its CSS class, you can give your TreeItems specific icons and such, in which case you need to change the CSS background-image for those specific MenuItems, which really is out of scope of this article.

130. Timer

Timer widget that will periodically call the server every n'th millisecond. You can set its period through the

131. Window

Window control. Basically an "advanced panel" with support for moving. It also features borders which can be skinned. The equivalent of a normal desktop window.

132. TreeView

This widget makes it possible for you to create selector kind of widgets that mimicks the representation of a tree kind-of hierarchical structure for displaying things. Useful for displaying for instance folder structures and such.

133. TextBox

A single-line type of 'give me some text input' type of widget. This widget is a wrapper around the input type="text" type of widget. If you need multiple lines of input, you should rather use the TextArea widget. However this widget is useful for cases when you need the user to give you one line of text input. See also the RichEdit widget if you need rich formatting of your text. This widget can also be set to 'password mode', which means whatever is typed into the widget will not be visible on the screen. Please notice though that by default, this will be transfered to the server in an unsecure manner, so this is only a mechanism to make sure that other people cannot read over your shoulder to see what you're 'secretly' trying to type into your TextBox. Use SSL or other types of security to actually implement safe transmitting of your passwords and similar 'secret text strings'.

134. TabButton

One instance of an item within a TabStrip. Mostly shown as buttons inside of the TabStrip.

135. SubMenu

Child control of a MenuItem. Will contain the SubMenuItems of a MenuItem. If you have MenuItems which have children, then you must create an item of type SubMenu and add those children inside of that object. This is true both for .ASPX markup and Menu hierarchies created in code.

136. TextArea

A multiple line type of 'give me some text input' type of widget. It wraps the textarea HTML element. If you only need single lines of input, you should probably rather use the TextBox widget. However this widget is useful for cases when you need multiple lines of text input. See also the RichEdit widget if you need rich formatting of your text.

137. TabStrip

A strip of TabButtons, used often to trigger changing active view in combination with the MultiPanel control. This control is a collection of TabButtons.

138. MenuItem

Items within a Menu. A MenuItem can have a Text property in addition to a SubMenu, which if defined will popup as a child of the MenuItem. The MenuItem class is basically the individual items of your menu.

139. CheckBox

A CheckBox is a 'two state button' which you can turn 'on' and 'off'. Useful for boolean UI situations where use must choose between two options, for instance 'yes' or 'no' situations.

140. HiddenField

Hidden field widget. Useful for having state you wish to pass on to the client but don't want it to be visible for the end user. Notice that this is not a safe place to put things that the user is not supposed to see, like passwords and such. Do NOT trust the value of this element to not be tampered with. Alternatives for using this widget is ViewState, cookies and the Session object. ViewState and cookies are neither 'safe' against tampering.

141. DynamicPanel

Control for making it easier to dynamically instantiate new controls and add them into your page. Completely abstracts away the entire hassle of storing whether or not a control has been previously loaded into the page or not. Make sure you handle the Reload event, and dependent upon the key given will load the exact same control and add it up into the DynamicPanel, and you should experience a very smooth experience in regards to dynamically loading controls into your page. Call the method LoadControl or AppendControl with a unique ID defining which control you wish to load, for instance the name of a UserControl file on disc, and until you explicitly clear your DynamicControl, the same control will be automatically loaded every time. If you do not create a Reload Event Handler for your widget, and you call LoadControl, then the DynamicControl will assume that what you're passing in is a fully qualifed path to a UserControl, and attempt to load it as such. You can use the Extra parameter to add extra initialization parameters into the control upon its first load.

142. AccordionView

Panels contained inside the Accordion control. Basically an Accordion is only a collection of these types of controls.

143. Accordion

Accordions are panels, or AccordionViews grouped together where you can expand and collapse individual AccordionViews within a group. Normally only one AccordionView can be visible at the same time, but this is configurable.

144. Button

A clickable button. The equivalent of input type="button". Use when you need a clickable thing to resemble a button. See also the LinkButton for an alternative. Also remember that any Widget in Magix UX can be made clickable, so you can also use a Label as your 'clickable thingie' if you wish. Even though anything can be made clickable in Magix UX, it is often an advantage to use buttons or link buttons since these elements will mostly be recognized by screen readers and such, and it is hence more 'polite' to use these specially designed types of 'clickable objects' such as the Button.

145. HyperLink

A wrapper around a hyper link or anchor HTML element (anchor HTML element ...) Sometimes you will need to create links that might change or needs changes after initially created. For such scenarios, this widget is highly useful.

146. LinkButton

This widget is another type of 'button widget', though this will be rendered using anchor HTML element (anchor element...) Even though everything can be made 'clickable' in Magix UX, it is definitely semantically much more 'correct' to constraint yourself to the ones that are expected to be 'clickable', such as this widget (LinkButton), Button, ImageButton etc. Among other things screen-readers and such will recognize these types of elements as 'clickable' and present the end user with the option of clicking these types of widgets.

147. Menu

Menu control. A Menu is basically a collection of MenuItems. Mimicks a Menu the way you're used to seeing them on desktop systems, or at other web applications. The Magix UX menu have support for any number of child menus, and are very customizable in regards to how it looks and behaves.

148. Label

A 'text widget'. The basic purpose of this widget is purely to display text and nothing else, though through the CssClass property and the Style property you can easily manipulate this to do mostly anything you wish. If a more 'complex widget' is needed, for instance to host other widgets, then the Panel widget is more appropriate to use than the Label. Unless the Tag property is changed, this widget will render as a span...

149. ImageButton

Technically this widget is completely redundant towards the Image Ajax Widget. But it is here for cases where you want to be semantically highly correct and need something that gives clues in regards to that it is 'clickable'. If you have an Image which is clickable, then semantically this widget is more correct to use.

150. Image

Image Ajax Widget. Useful for showing images that needs Ajax functionality somehow. Notice that most times it's more efficient to display other types of widgets, such as the Panel or a Label and set it to display an image through using something such as background-image through CSS or something similar.

151. CompositeControl

Abstract helper widget for widget developers. Used internally in e.g. Window and Accordion. Useful for those cases where you have a widget which is 'composed' out of both a norml 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself. The Window is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Caption and a Close button. That's why the Window is a 'CompositeControl'.

152. BaseWebControlFormElementText

Abstract base class for WebControls which are BaseWebControlFormElement type of controls, but also have a Text property. Text property is overridable, its default implementation sets the 'Value' property on the client-side, which may or may not be suitable for your needs.

153. ViewCollectionControl-g

Abstract base class for widgets which are supposed to contain only one type of child widgets. Contains a lot of usable shorthands like for instance Enumerable support, and an index operator overload.

154. CompositeViewCollectionControl-g

Abstract helper widget for widget developers. Used internally in e.g. TreeItem. Useful for those cases where you have a widget which is 'composed' out of both a norml 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself, and you in addition to this also have a specific type of widgets that are supposed to be its child control collections, like for instance the TreeItem are only supposed to have TreeItems within itself. The TreeItem of the TreeView is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Text property and many other child controls. In addition it is supposed to only have child controls of type TreeItem. That's why the TreeItem is a 'CompositeViewCollectionControl'.

155. BaseWebControl

MUX WebControl equivalent. Contains a couple of additions to the MuxBaseControl. Among others a style attribute and a CssClass. Also contains most of all the DOM event handlers you can possibly handle in Magix UX. In addition to most of the client-side effects you can possibly run on your MUX controls. Class is abstract, and hence cannot be instantiated directly but is meant for inheriting from.

156. BaseControl

Abstract base class for mostly all widgets in Magix UX. This is where the Ajax 'core engine' to a large extent exists within Magix UX. Contains several interesting features, such as for instance the Info property, which can take any string information and serialize back and forth between server-requests. If you need to create your own Ajx Control, you should either directly, or most probably indirectly, inherit from this MuxBaseControl. If you inherit, directly or indirectly, form this class you need to override several methods from this class. The most notable is RenderMuxControl. If you have a 'visual widget', you will mostly inherit from MuxBaseWebControl instead of this class. If your control is 'non-visual', such as the MUX Timer, you will mostly inherit from this class.

157. BaseWebControlFormElementInputText

Abstract base class for widgets being BaseWebControlFormElementText type of controls, but also uses the Text portions as an input value which the user can change himself through interacting with the widget.

158. BaseWebControlFormElement

Abstract base class for widgets which are HTML FORM type of elements. Abstracts away things such as blur and focus event handling, enabled, keyboard shortcuts and other internals things. Inherit from this one if you intend to create an Ajax Widget which wraps an HTML FORM element.

1. Welcome to Magix!

Truly a Strange and Wonderful World ...

A place where you can become literate in regards to computers. A place where you can express yourself, creating what you want out of your computer. A place where you are in charge, a place which is fun!

This is your First tip of the day. Leave These Tips On to make sure you get Useful Tips and Tricks as you proceed deeper and deeper into the Rabbit Hole ...

In fact, your first Tip is to use the Arrow Keys in the Top/Right corner of this window to navigate forward and fast read the next upcoming 5-10 tips. They are all crucial for getting started with Magix ...

If you need to re-read a previous tip, you can click the previous button ...

4 sec Intro

Magix consists of two majorly important pre-installed modules; The 'Publishing' System and the 'MetaType' System.

Publishing is where you go to create your WebSite, and MetaTypes is where you go to create Applications. Though, really the difference is more blurry than you think. For instance; It is difficult to show any Applications to your end users, if you do not have any Pages to 'host' your Application ...

So Pages can be thought of as Views in your Applications if you want to, or your entire hierarchy of Pages can be viewed as a gigantic plugin Application, which it actually in fact is ... ;)

It is actually quite useful to stop separating between 'Old-Time Constructs' such as 'code', 'data', 'input' and 'output'.

Old World thinking, trying to categorize things into different types, are much less useful in Magix than what you think. In Magix, everything is kind of 'mushy'. Or 'everything is everything' I guess you can say. This is what makes it possible for you to Stay in Control and deliver Secure and Stable Systems, regardless of the Complexity of your Domain Problem ...

We recommend people to Start with Learning Publishing and how the WebPages work. Then later, only when a firm grasp of Pages and Templates are understood, we recommend moving onto Applications ...

Basics ...

But before we can do anything else, we need to learn the Basics ...

Most of Magix is made up of 'Basic Components', which are tied together to create a whole.

For instance, a button will mostly look the same everywhere. By default a button will be Gray and use Bold, Black and Big Fonts ...

A good example of a Button is the Top/Right corner of this tooltip, which has two Buttons. One Paging forward, and another paging Backwards in the Hierarchy of Tips and Tricks ...

A 'Grid' is when you see a list of items. A good example of a Grid would be MetaTypes/Meta Actions ...

There are many types of 'Basic Components' like these in Magix. Over the next couple of pages, we'll be walking through some of them which you'll need to understand to be able to get the most out of Magix ...

Grids

Most Grids in Magix have tons of features. These features includes; Paging, In Place Editing of Values, Filtering, and so on ...

To Filter according to a Column, all you've got to do is to click the header of your Grid, choose which type of Filter you want to apply, type in its value, and click OK ...

The arrow buttons, normally at the bottom of your grids makes it possible to traverse forward and backwards in your list of items. The double arrows takes you 'all the way' in its direction ...

Open up Meta Types/Meta Actions and play around with that grid by filtering, creating a couple of new items and so on.

Make sure you don't change any of the existing items, since some things are dependent upon 'System Actions' which must be defined for your system to properly work ...!

To see Paging you'll normally need to have more than 10 items in your grid. To see 'all the way paging', you'll normally need more than 20 items ...

Make sure you also click the 'Edit' column. Sometimes this column will say 'Edit' while sometimes it'll show a number like in the Action view. However, clicking the Edit Column, will always somehow bring you to a View where that object can be edited in 'full version' ...

Some things should be very similar for mostly all grids like this in Magix. For instance ...

Clicking the '+' button will almost always create a new object of that type ...

If the Text of a Grid Cell is Blue, this means that you can edit the value directly by clicking the Blue Text, which will exchange it with a 'textbox', from where you can edit its value ...

Publishing ...

A Website consists of Pages. Every Page is the equivalent of one 'URL'. Although URLs doesn't really exist in Magix, it helps to think of a page as such. Beside, creating a URL based Navigation Plugin would be piece of cake anyway, due to the Architectural Principles Magix is built on ...

Anyway ...

You create your pages according to 'Templates', which can be seen as 'Recipes' for your pages. You have to have at least One Template in your system before you can start creating Pages. Every Page is based upon a Template, and no page can exist without its Template ...

Click 'Publishing->Templates ...' now!

If you click 'Dashboard' at the root of your menu you will return back here. To access the root of your menu, click the left arrow at the top of your Sliding Menu ...

PS!Click the Edit links to view any specific templates ...

Every Template contains a bunch of WebPart Templates. These WebPart Templates can be positioned exactly as you wish on your page. When editing a Template, use the Arrow Buttons to position your WebParts ...

Go check out 'Publishing->Templates ...' one more time, and see how you can move stuff around on your 'surface' by clicking the Arrow Buttons ...

When positioning your WebPart Templates, realize that you're not really 'positioning' them, but rather you are changing their width, height and margins. In the beginning this might feel a little bit cumbersome, though after some time you'll hopefully appreciate this 'floating layout' and become used to it ...

Realize also that especially the bottom and right margins might create funny looking WebParts since they're not really visible while editing. If you're having weird results, make sure your right and bottom margins are 0 by double clicking them, which should set them back to zero ... ;)

Double clicking any of the arrows will either maximize or minimize their associated property ...

A WebPage Template must contain at least one WebPart Template. A WebPart Template is also a 'Type Definition' for your WebParts. WebPart Templates have names such as 'Content' and 'Header', which are publishing modules for showing large letters and rich text fragments. Every WebPart Template is based upon one plugin type.

Meaning if you have one page, based upon a WebPage Template, with 5 WebPart Templates, you'll have a WebPage with 5 WebParts where each WebPart can be different 'Applications'. This might be any combination of Applications, such as Text Fragments, Headers, CRUD Forms and such ...

However, we'll stick to 'Publishing' as we promised in the beginning, and focus on the Publishing Modules ...

If you edit the default Template created by the system for you, you can see how it has Menu, Header and Content as 'Module Names'

Go check it out while I hang around here ...

The Menu is similar to the Sliding Menu to the left which you're using yourself, while the Header will show an H1 HTML element [Header Element] and the Content module will show Rich Editable Content.

If you have more types of Modules in your installation of Magix, these might also show up as selections in the DropDownBoxes visible while editing your Templates ...

No go to Templates and change the Name of Template 'M+H+C'. Change it to 'Testing'. This can be done by clicking directly on the text where it says 'M+H+C'. Then open up 'Pages ...' and click your root page.

Do you see how the name of the module in the DropDown box, roughly at the middle of the screen has changed now to 'Testing'. This is because that DropDown box is being used to select a Template for your WebPage.

Then try to change the Name of your Templates by clicking e.g. Header while editing your Template, and type in 'Header2'. Now edit your Page and see how this change reflects from the Template and to the Page.

Go take a look, while I chill ...

Important: If you change the number of WebParts in your Template, or you change the type of Module of your WebPart Template, then all pages built upon that Template will have to be resaved, and you'll probably loose data.

It is therefor important that you create your templates first, and then don't edit these two properties while they're already in 'Production' ...

Try to Create several different Templates, and have slightly different values for their width, height, margins and such.

Be certain of that you've added different widths of your Menu Containers and different Left Margins

Make sure they've got the same type of modules in the same container

Then use these different Templates for different pages which you create in your Pages hierarchy

If you now access the root of your website, and try to browse around by clicking different buttons, you can see how the WebPart Containers are 'jumping around' on the screen ...

Init. of WebParts

WebParts initializes differently depending upon where you're coming from, and from which Template type you're coming from dependent upon to which Template type you're entering. And also according to which type of WebParts, or Modules they are ...

For instance the sliding menu will not reload as long as the container it is within on one page template, is the same container it is within on the next page template ...

Too confusing ...?

Just remember; always have a Menu positioned in the same container [first one for instance?] on all of your templates, unless you really know what you're doing ...

Play around with the system by creating some new Templates, copying them, change their weparts type between Header, Content and SliderMenu. Then when you come back, we'll start diving into Applications ... ;)

Applications ...

Most applications will be WebParts. This means that you can inject them into any WebPart onto any WebPage you wish.

Let's create an Application ... :)

First make sure you have one Template in use in one of your pages which has one WebPart Template with the Module Type of 'MetaView_Single' ...

Then click on 'MetaTypes/Meta Views' ...

Create a new View called 'CollectEmails' ...

Add three properties to your form, name them

\tName\tEmail\tSubscribe

Change their description to something meaningful ...

Make sure you change the 'Type Name' of your object to 'EmailSubscription'

Attach two Actions to your 'Save' property.

\tMagix.DynamicEvent.SaveActiveForm\tMagix.DynamicEvent.EmptyActiveForm

The first Action will save your form, while the second one will empty it.

Now try to View your form in preview mode, and test it out by typing in your email and name, and clicking Submit to save your Object ...

PS!Obviously it's crucial that the 'Save' action runs before the 'Empty' action, in case you wondered ... ;)

TypeNames ...

If you take a look at your Meta Objects now you will see a new object with the Type Name of 'EmailSubscription'. The 'Type Name' property from your MetaView decides the Type Name of your Objects ...

These 'Type Names' are important to distinguish from different types of Objects.

One Object might be of type 'Customer', while another object might be of type 'Email', and so on. What names you give your Types is crucial! Name clashes here might create very hard to track down bugs and such ...

Take some care when naming your Types!

It's probably a good practice to some how make sure they've got unique names, also across your organization if you want to use plugins made by others.

We encourage people to use type names such as; "CompanyName.Department.Customer", and never 'Customer' directly. In fact, your homework for this lesson is to go and rename your 'Customer' TypeName, and rename the Type Name to; 'CompanyName.Department.Customer'. Where Company Name and Department are your company name and your department ...

OpenID

Did you know that you can use Magix as both a Relying Party and OpenID Provider?

[Read more about Open ID here ...]

If you need an OpenID token to log into some website somewhere, then you can append ?openID=admin after the root URL to your website, if the admin user is the User you'd like to log in with.

[Full example; yourdomain.com/?openID=admin]

This will redirect from the website you're trying to log into, and back to your website, which in turn will ask you for admin's password.

Once successfully logged into your own Magix website, your website will redirect back to the website you're trying to log into, and tell it that it 'has prof of that you are who you claimed you were'.

You can also associate other OpenID tokens, such as your Yahoo account or Blogger account, with your Magix User internally. This will allow you to log into Magix with that OpenID Account.

If your blogger account is 'magix' for instance, then your OpenID 'username' [claim] would become 'magix.blogspot.com'

Talk to CEO

If you make something cool with Magix, that you want to share, then we'd love to get to know about it. It can be an instructional YouTube video about how to Get Started, Tips and Tricks etc. It can be a Tutorial Blog you've written about how to install Magix on your server. It can be a book you have written about the O2 Architecture. It can be an Action, or a collection of Actions, which you think is awesome. Anything really!

As long as it has Value for our Community somehow, we'd love to know about it, so that we could help you promote it, and more importantly; Helping our Community out ... :)

Our CEO's Name and Email address is; Lissa Millspaugh - lissa.millspaugh@winergyinc.com

Our CTO's Name and Email address is; Thomas Hansen - thomas.hansen@winergyinc.com

If your stuff is of 'geeky nature', it's probably best to send it to our CTO, or at least CC him in ... ;)

2. Meta Actions

Meta Actions, or 'Actions' for short, are dynamically created actions within the system, either created by the system itself during installation or something, or Actions createdby users, such as you.

These Actions can then later be invoked by some sort of event, for instance a user clicking a button, or something similar

Basically, if it can be described as a 'verb', it's probably an Action ... ;)

SaveActiveForm

Magix.DynamicEvent.SaveActiveForm

Magix.MetaView.CreateSingleViewMetaObject

Will save the currently active Single-View Form.Will determine which form raised the event originally, and explicitly save the field valuesfrom that Form into a new Meta Object with the TypeName from the View ...

CreateGallery

Magix.DynamicEvent.CreateGallery

Magix.Common.CreateGallery

Will create a Gallery object from the given 'Files' list within the 'Folder'

IncludeCSSFile

Magix.DynamicEvent.IncludeCSSFile

Magix.Core.AddCustomCssFile

Will include a CSS file onto the page, even in an Ajax Callback if you wish. Change the 'CSSFile' parameter to choose which CSS file you'd like to include

PlaySound

Magix.DynamicEvent.PlaySound

Magix.Core.PlaySound

Will play the given 'File' sound. If the sample sound file doesn't work, you've probably got something wrong with the setup of your Web Server. Make sure the file extension .ogg is associated with the MIME type of audio/ogg. PS! For the record; Cool-Breeze.ogg and The-Last-Barfly.ogg are both songs composed by our CTO Thomas Hansen. They are both performed by Thomas Hansen and his wife Inger Hoeoeg, and are to be considered licensed to you under the terms of Creative Commons Attribution-ShareAlike 3.0

CreateQRCode

Magix.DynamicEvent.CreateQRCode

Magix.QRCodes.CreateQRCode

Will create a QR Code with the given 'FileName' path and filename, which should end with .png. The QR Code will point to the given 'URL', and it will use the textures of 'BGImage' and 'FGImage' to render the code. The QR Code will have 'RoundedCorners' radius of rounded corners, and it will be 'AntiPixelated', and have the descriptive text of 'Text'. The QR Code will use 'Scale' number of pixels per square to render

PauseSound

Magix.DynamicEvent.PauseSound

Magix.Core.PauseSound

Stops Any sound or music currently being played

ResumeSound

Magix.DynamicEvent.ResumeSound

Magix.Core.ResumeSound

Resumes any sound or music previously being halted through 'StopSound' or other similar mechanisms. PS! Will throw exception if no sounds have been played, and hence no resuming can occur in any ways

EmptyActiveForm

Magix.DynamicEvent.EmptyActiveForm

Magix.Meta.Actions.EmptyForm

Will empty the currrently active Editable Form. Will determine which form raised the event originally, and explicitly empty that form only. Useful for things such as 'Clear Buttons' and such ...

RedirectClient

Magix.DynamicEvent.RedirectClient

Magix.Common.RedirectClient

Will redirect the client's browser to thegiven URL parameter

ImportCSVFile

Magix.DynamicEvent.ImportCSVFile

Magix.Common.ImportCSVFile

Will import the given 'FileName' from the given 'Folder' and transform to MetaObjects using the given 'MetaViewName'

ShowDefaultMessage

Magix.DynamicEvent.ShowDefaultMessage

Magix.Core.ShowMessage

Will show a default message to the User. Mostly here for Reference Reasons so that you can have an Example Action to copy for your own messages. For your convenience ... :)

TurnOnDebugging

Magix.DynamicEvent.TurnOnDebugging

Magix.Common.SetSessionVariable

Will turn on 'Debugging', meaning you'll have a wire-gridcovering your screen to see the 40x18 pixel 'grid-lock', plus you'll also get to see every single Action ever raised on the server shown in an 'Action Stack Trace' Window. This only affects your session, meaning it should be safe to do in production to track down errors and such in live software ...

TurnOffDebugging

Magix.DynamicEvent.TurnOffDebugging

Magix.Common.SetSessionVariable

Will turn _OFF_ 'Debugging', meaning you'll no longer have a wire-grid covering your screen, plus the stack tracing of actions on the server will disappear. Only affects your session, and no other logged on users ability to see debugging information ...

ViewMetaViewMultiMode

Magix.DynamicEvent.ViewMetaViewMultiMode

Magix.MetaType.ViewMetaViewMultiMode

Will load a grid of all Meta Objects of type already loadedin current activating WebPart. If you want to load a specific type, then you can override the type being loaded by adding 'MetaViewTypeName' as a parameter, containing the name of the view. There are many other properties you can override...

SendEmail

Magix.DynamicEvent.SendEmail

Magix.Common.SendEmail

Will send yourself an email to the Email address you've associatedwith your user. The email will contain a default header and a default body. Override the settings if you wish to send other emails, to other recipes, with another subject and/or body. PS! This Action is dependent upon that you've configured your web.config to point towards a valid 'mailSettings'.You _CANNOT_ use rasoftwarefactory.com for this! You might however be able to use for instance your Google Account if you do some 'Googling' ... ;)

ReplaceStringValue

Magix.DynamicEvent.ReplaceStringValue

Magix.Common.ReplaceStringValue

Will transform every entity of 'OldString' found in 'Source' into the contents of 'NewString' and return as a 'Result', output node ...

MultiAction

Magix.DynamicEvent.MultiAction

Magix.Common.MultiAction

Will raise several Actions consecutively, in the order they're definedin the 'Actions' node. Each Action needs a 'Name' and its own set of parameters through its 'Params' node.All 'Params' nodes will be copied into the root node before every event is raised. This means that yourRoot node will become VERY large after subsequent actions. Be warned ...

GetObjectIntoNode

Magix.DynamicEvent.GetObjectIntoNode

Magix.Common.GetSingleMetaObject

Will put every property from the given Meta Object, into the given Node, with the name/value pair as the node name/value parts, assuming they're all strings. Copy this Action, and make sure you _CHANGE_ its MetaObjectIDtowards pointing to the ID of a real existing Meta Object, to fill in values from one ofyour Meta Objects into a Node, maybe before sending the node into another even, usingMultiActions or something ...

SetMetaObjectValue

Magix.DynamicEvent.SetMetaObjectValue

Magix.MetaType.SetMetaObjectValue

Will set the value of the given MetaObject [MetaObjectID] to the Value of your 'Value' node at the 'Name' property.

GetActiveFormData

Magix.DynamicEvent.GetActiveFormData

Magix.MetaView.SerializeSingleViewForm

Will find the Form that raised the current eventchain, and query it to put all its data flat out into the current Node structurewith the Name/Value as the Node Name/Value pair.

LoadSignatureModule

Magix.DynamicEvent.LoadSignatureModule

Magix.Common.LoadSignatureForCurrentMetaObject

Will load the Signature Module inwhatever container its being raised from. And set the 'Value' property of the givenMetaObject Column Property Name to the signature signed on the Signature module ...

ExportMetaView2CSV

Magix.DynamicEvent.ExportMetaView2CSV

Magix.Common.ExportMetaView2CSV

Will render the 'Currently Viewed' MetaView intoa CSV file [Microsoft Excel or Apple Numbers etc] and redirect the users client [Web Browser] to the newly rendered CSV file. 'Currently Viewed' meaning the view that contained the control that initiated this Action somehow. If you explicitly create a 'MetaViewName' parameter, and set its name to another MetaView, then that MetaView will be rendered instead

3. MainWebPage

Magix.Brix.ApplicationPool.MainWebPage

Description:
Level3: Your 'Application Pool', meaning the 'world' where all your 'components' lives. Nothing really to see here, this should just 'work'. But are here for reference reasons

4. Magix_PageStatePersister

Magix.Brix.ApplicationPool.Magix_PageStatePersister

Description:
Level3: Implements serialization of ViewState into the database on the server side

Methods

public override void Load()

Level3: Loads Viewstate from database

public override void Save()

Level3: Saves Viewstate to database

5. TextAreaEdit

Magix.Brix.Components.TextAreaEdit

Description:
Level4: Basically every single instance you see in Magix which is of type 'InPlaceEditable' is an instance of this component. Alows for viewing textual based information, while at the same time, this tet can be 'clicked' which will change it into a text box, in which you can change its value. Often displayed with a pen next to it

Basetypes
Magix.UX.Widgets.Panel

Events

public EventHandler TextChanged
Level4: Text was changed by end user

public EventHandler DisplayTextBox
Level4: Raised when TetBox is about to become displayed

Properties

public string Text

Level4: The actual text property of the control

public int TextLength

Level4: Max length before 'clipping' will occur

6. Module

Magix.Brix.Components.Module

Description:
Level4: Baseclass for the 'Edit Objects' parts of the Grid system in Magix. Not meant for directly consuming through this class

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

protected void Magix_Core_UpdateGrids(object sender, ActiveEventArgs e)

Level4: Handled to make sure we update ours too, since object might be changed some of its properties

protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, ActiveEventArgs e)

Level4: Checking to see if it may be us, and if so, re-databind

protected void RefreshWindowContent(object sender, ActiveEventArgs e)

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

7. ListModule

Magix.Brix.Components.ListModule

Description:
Level4: Implements most of the logic for our DBAdmin module and basically every single Grid we have in Magix. This class is the foundation for probably most Magix screens you've ever seen, since every Grid within the system, virtually, is built using this class. Have so many bells and whistles, it could probably need its own book, not to mention some serious refactoring. Till at least the refactoring parts are over, I think I'll resist the temptation of going 'over the board' in documenting this, since first of all everything will change later, and you shouldn't fiddle too much directly with this bugger yourself. There's also PLENTY code samples around ANYWHERE in magix of usage of this class, so have fun :)

Basetypes
Magix.Brix.Components.Module

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

public void Magix_Core_SetGridPageStart(object sender, ActiveEventArgs e)

Level4: Will 'page' back to 'start' [whatever that is]

public void DBAdmin_Grid_SetActiveRow(object sender, ActiveEventArgs e)

Level4: Will set the Active Row of the grid, if the 'FullTypeName' is correct

protected abstract void DataBindDone()

Level4: Called when databinding are done

Properties

protected abstract ControlTableParent

Level4: The control being our 'table element'

8. DBAdmin_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.DBAdmin.DBAdmin_Controller

Description:
Level2: Contains the logic for the DBAdmin module(s), which is also the Grid/CRUD-Foundation system in Magix. Contains many useful methods and ActiveEvents for displaying either Grids or editing Single Instances of Objects. Can react 100% transparently on ActiveTypes. Has support for Meta Types, meaning types where you're more in 'control', but must do more of the 'arm wrestling directly'

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="DBAdmin.Form.ViewClasses")]
protected void DBAdmin_Form_ViewClasses(object sender, ActiveEventArgs e)
```

Level2: Loads up the DBAdmin BrowserClasses interface. Pass in a node for positioning and such

```
[ActiveEvent(Name="DBAdmin.Data.GetClassHierarchy")]
protected void DBAdmin_Data_GetClassHierarchy(object sender, ActiveEventArgs e)
```

Level2: Will return the Class Hierarchy of all ActiveTypes within the system in a tree hierarchy, according to namespace

```
[ActiveEvent(Name="DBAdmin.Form.ViewClass")]
protected void DBAdmin_Form_ViewClass(object sender, ActiveEventArgs e)
```

Level2: Will show all the objects of type 'FullTypeName' by using the passed in node for settings in regards to positioning and such

```
[ActiveEvent(Name="DBAdmin.Data.GetContentsOfClass")]
protected void DBAdmin_Data_GetContentsOfClass(object sender, ActiveEventArgs e)
```

Level2: Will return a range of objects type 'FullTypeName' depending upon the 'Start' and 'End' parameter. Requires a "FullTypeName" parameter, and "Start" + "End"

```
[ActiveEvent(Name="DBAdmin.Data.ChangeSimplePropertyValue")]
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, ActiveEventArgs e)
```

Level2: Will change a 'Simple Property Value' [property of some sort belonging to the object, e.g. a DateTime/Birthday property] Needs 'FullTypeName', 'ID', 'PropertyName', 'NewValue' to work. NewValue must be of type of property.

```
[ActiveEvent(Name="DBAdmin.Form.ViewListOrComplexPropertyValue")]
protected void DBAdmin_Form_ViewListOrComplexPropertyValue(object sender,
ActiveEventArgs e)
```

Level2: Will show either a Child object or a List of children depending upon the 'IsList' parameter

```
[ActiveEvent(Name="DBAdmin.Data.GetListFromObject")]
protected void DBAdmin_UpdateComplexValue(object sender, ActiveEventArgs e)
```

Level2: Returns a Range of Child Objects belonging to the 'ParentID'. Needs 'ParentFullTypeName', 'ParentPropertyName', 'Start', 'End' and 'ParentID' to function

```
[ActiveEvent(Name="DBAdmin.Data.GetObject")]
protected void DBAdmin_Data_GetObject(object sender, ActiveEventArgs e)
```

Level2: Will return the object with the given 'ID' being of 'FullTypeName' as a Value/Key pair

```
[ActiveEvent(Name="DBAdmin.Data.GetObjectFromParentProperty")]
protected void DBAdmin_Data_GetObjectFromParentProperty(object sender, ActiveEventArgs e)
```

Level2: Will return a single instance of a complex [ActiveType normally, unless 'meta'] child object

```
[ActiveEvent(Name="DBAdmin.Form.ViewComplexObject")]
protected void DBAdmin_ShowComplexObject(object sender, ActiveEventArgs e)
```

Level2: Will show one Complex object with the 'ID' and 'FullTypeName'

```
[ActiveEvent(Name="DBAdmin.Form.ConfigureFilterForColumn")]
protected void DBAdmin_Form_ConfigureFilterForColumn(object sender, ActiveEventArgs e)
```

Level2: Will open up a 'Configure Filter' dialogue from which the user can change, edit or remove any existing filters or create new ones

```
[ActiveEvent(Name="DBAdmin.Form.ShowAddRemoveColumns")]
protected void DBAdmin_Form_ShowAddRemoveColumns(object sender, ActiveEventArgs e)
```

Level2: Will load up 'Configure Columns to View form' to end user

```
[ActiveEvent(Name="DBAdmin.Data.ChangeVisibilityOfColumn")]
protected void DBAdmin_Data_ChangeVisibilityOfColumn(object sender, ActiveEventArgs e)
```

Level2: Changes the visibility setting of a specific Column for a specific type

```
[ActiveEvent(Name="DBAdmin.Data.DeleteObject")]
protected void DBAdmin_Data_DeleteObject(object sender, ActiveEventArgs e)
```

Level2: Will delete the given 'ID' ActiveType within the 'FullTypeName' namespace/name after user has been asked to confirm deletion

```
[ActiveEvent(Name="DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Data_ComplexInstanceDeletedConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Default implementation of 'deletion of object was confirmed by user' logic

```
[ActiveEvent(Name="DBAdmin.Common.ComplexInstanceDeletedNotConfirmed")]
protected void DBAdmin_Data_ComplexInstanceDeletedNotConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Flushes the Container containing the MessageBox

```
[ActiveEvent(Name="DBAdmin.Common.CreateObject")]
protected void DBAdmin_Common_CreateObject(object sender, ActiveEventArgs e)
```

Level2: Will create a new object of type 'FullTypeName'

```
[ActiveEvent(Name="DBAdmin.Common.CreateObjectAsChild")]
protected void DBAdmin_Common_CreateObjectAsChild(object sender, ActiveEventArgs e)
```

Level2: Will create a new object of type 'ParentFullTypeName' and append it to the 'ParentID' 'ParentPropertyName' property which must be of type 'FullTypeName'

```
[ActiveEvent(Name="DBAdmin.Form.AppendObject")]
protected void DBAdmin_Form_AppendObject(object sender, ActiveEventArgs e)
```

Level2: Will show a list of objects of type 'FullTypeName' and allow the user to pick one to append into 'ParentID' 'ParentPropertyName' with the given 'ParentFullTypeName'

```
[ActiveEvent(Name="DBAdmin.Data.AppendObjectToParentPropertyList")]
protected void DBAdmin_Data_AppendObjectToParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Will append an object to a list of objects in 'ParentID' ParentPropertyName collection and save the 'ParentID' object

```
[ActiveEvent(Name="DBAdmin.Data.ChangeObjectReference")]
protected void DBAdmin_Data_ChangeObjectReference(object sender, ActiveEventArgs e)
```

Level2: Will change a single instance object reference between 'ParentID' and 'ID' in the 'ParentPropertyName' of 'ParentFullTypeName'. Flushes child container

```
[ActiveEvent(Name="DBAdmin.Form.RemoveObjectFromParentPropertyList")]
protected void DBAdmin_Form_RemoveObjectFromParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Removes a referenced object without deleting it [taking it out of its parent collection] Notice that if the Parent object is the 'Owner' of the object, it may still be deleted. Will ask for confirmation from end user before operation is performed

```
[ActiveEvent(Name="DBAdmin.Form.RemoveObjectFromParentPropertyList-Confirmed")]
protected void DBAdmin_Data_RemoveObjectFromParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Removes an object out of its 'ParentID' 'ParentPropertyName' collection of type 'ParentFullTypeName'

```
[ActiveEvent(Name="DBAdmin.Form.ChangeObject")]
protected void DBAdmin_Form_ChangeObject(object sender, ActiveEventArgs e)
```

Level2: Will show the 'Change Single-Object Reference' form to the end user

```
[ActiveEvent(Name="DBAdmin.Data.RemoveObject")]
protected void DBAdmin_Form_RemoveObject(object sender, ActiveEventArgs e)
```

Level2: Removes a single-object reference from the 'ParentID' object

```
[ActiveEvent(Name="DBAdmin.Data.GetFilter")]
protected void DBAdmin_Data_GetFilter(object sender, ActiveEventArgs e)
```

Level2: Returns the filters for different columns in the Grid system

```
[ActiveEvent(Name="DBAdmin.Data.SetFilter")]
protected void DBAdmin_Data_SetFilter(object sender, ActiveEventArgs e)
```

Level2: Changes the filter for a specific 'Key'/'Value' for a specific type

9. ColumnTypesPlugins_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.DBAdmin.ColumnTypesPlugins_Controller

Description:
Level3: Contains some template columns for the Grid system for you to use in your own Grids

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.DataPlugins.GetTemplateColumns.CheckBox")]
protected void Magix_DataPlugins_GetTemplateColumns_CheckBox(object sender,
ActiveEventArgs e)
```

Level3: Creates a CheckBox type of column for the Grid System

10. Documentation_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Documentation.Documentation_Controller

Description:
Level2: Contains the logic for our 'Class Browser' which can browse all the classes in the system and make some changes to them by enabling them and disabling them by either overriding specific events or by disabling entire controllers or modules all together

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, ActiveEventArgs e)
```

Level2: Will return the menu items needed to start the class browser

```
[ActiveEvent(Name="Magix.MetaType.ViewDoxygenFiles")]
protected void Magix_MetaType_ViewDoxygenFiles(object sender, ActiveEventArgs e)
```

Level2: Gets every namespace and class into a tree hierarchy and loads up the Tree module to display it

```
[ActiveEvent(Name="Magix.Doxygen.Nei!Det_E_Boka_Mi ...!")]
protected void Magix_Doxygen_Give_Me_My_Bok_Mann(object sender, ActiveEventArgs e)
```

Will generate the PDF containing the entire documentation, plus potential plugins documentation features, such as the MetaAction and MetaView system etc. Will redirect the client to download the PDF file once generated.

```
[ActiveEvent(Name="Magix.Doxygen.RollLevel")]
protected void Magix_Doxygen_RollLevel(object sender, ActiveEventArgs e)
```

Rolls level on documentation advanceness one up, til 4, and then back to zero. Reloads Doxygen Documentation class/namespace Tree afterwards

```
[ActiveEvent(Name="Magix.Doxygen.ViewNamespaceOrClass")]
protected void Magix_Doxygen_ViewNamespaceOrClass(object sender, ActiveEventArgs e)
```

Level2: Expects a SelectedItemID which should point to either a Namespace or a Class, and will show this namespace/class features. Such as which dll(s) implements the namespace/class, if a class, which methods it has etc

```
[ActiveEvent(Name="DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure we handle the "Documentation_Controller-META" type for the Grid system so that we can see our DLLs

```
[ActiveEvent(Name="DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure we handle the "Documentation_Controller-META" type for the Grid system so that we can see our DLLs

Properties

```
public  Docs  Docs
```

Will cache our Doxygen.NET objects, such that they're faster available

11. Email_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Email.Email_Controller

Description:
Level2: Implements logic for sending email using SMTP through the .Net classes

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Async=true, Name="Magix.Core.SendEmail")]
protected void Magix_Core_SendEmail(object sender, ActiveEventArgs e)
```

Level2: Will send an Email, using the SMPT settings from web.config, with the given Header and Body to the list in the EmailAddresses parameter from the AdminEmail and AdminEmailFrom parameters

12. FileExplorer_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.FileExplorer.FileExplorer_Controller

Description:
Level2: Contains Logic for file explorer. The file explorer is a component where you can browse the file folder system on your web server, remotely, and do many operations. Such as editing CSS files, uploading images and such. PS! To use this Module it is IMPERATIVE that you've given the 'NETWORK SERVICE' account 'Full Access' to at the very least the 'media/' folder, or whatever folder you plan to use the explorer on on your web server

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.FileExplorer.LaunchExplorer")]
protected void Magix_FileExplorer_LaunchExplorer(object sender, ActiveEventArgs e)
```

Level2: Use this method to launch the file explorer. You can override any parameter you wish. The default file filter is; "*.png;*.jpeg;*.jpg;*.gif;" but can be changed through the "Filter" parameter. If you add up CSS, and set "CanCreateNewCssFile" to true, then CSS files can both be edited and created on the fly. If you set "IsSelect", then the end user can "select" files, which upon selection is done will raise the 'SelectEvent'

```
[ActiveEvent(Name="Magix.FileExplorer.GetFilesFromFolder")]
protected void Magix_FileExplorer_GetFilesFromFolder(object sender, ActiveEventArgs e)
```

Level2: Will retrieve the 'FolderToOpen' folder's files back to caller

```
[ActiveEvent(Name="Magix.FileExplorer.EditAsciiFile")]
protected void Magix_FileExplorer_EditAsciiFile(object sender, ActiveEventArgs e)
```

Level2: Will open up EitAsciiFile 'Notepad'ish' Editor with the given 'File' for editing. Intended to allow editing of CSS files and other types of ASCII files

```
[ActiveEvent(Name="Magix.FileExplorer.FileSelected")]
protected void Magix_FileExplorer_FileSelected(object sender, ActiveEventArgs e)
```

Level2: Will retrieve the properties for the file, such as disc size, width/height if image, etc

```
[ActiveEvent(Name="Magix.FileExplorer.ChangeFileName")]
protected void Magix_FileExplorer_ChangeFileName(object sender, ActiveEventArgs e)
```

Level2: Changes the name of the file with 'OldName' to 'NewName' within the given 'Folder'

```
[ActiveEvent(Name="Magix.FileExplorer.DeleteFile")]
protected void Magix_FileExplorer_DeleteFile(object sender, ActiveEventArgs e)
```

Level2: Will delete the 'File' within the 'Folder' of the system

13. Logger_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Logger.Logger_Controller

Description:
Level2: Contains logic for Logging things

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Core.UserLoggedIn")]
private void Magix_Core_UserLoggedIn(object sender, ActiveEventArgs e)
```

Level2: Handled here since it's one of the more 'common operations' and probably interesting to see in retrospect in case something goes wrong

```
[ActiveEvent(Name="Magix.Core.ShowMessage")]
protected void Magix_Core_ShowMessage(object sender, ActiveEventArgs e)
```

Level2: Handled here since it's one of the more 'common operations' and probably interesting to see in retrospect in case something goes wrong

```
[ActiveEvent(Name="Magix.Core.Log")]
protected void Magix_Core_Log(object sender, ActiveEventArgs e)
```

Level2: Will create one LogItem with the given LogItemType, Header, Message, ObjectID, ParentID, StackTrace and so on, depending upon which data is actually being passed into it. Minimu requirement is 'Header'

```
[ActiveEvent(Name="Magix.Core.NewUserIDCookieCreated")]
protected void Magix_Core_NewUserIDCookieCreated(object sender, ActiveEventArgs e)
```

Level2: Handled since it's important information since it's highly likely it's a new visitor to the app/website

14. MetaAction_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.MetaAction_Controller

Description:
Level2: Contains logic for Actions which are wrappers around ActiveEvents for the end user to be able to raise his own events, with his own data in the Node structure. The MetaAction system is at the core of the Meta Application system wince without it the end user cannot create his own types of events or Actions

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, ActiveEventArgs e)
```

Level2: Will return the menu items needed to fire up 'View Meta Actions' forms for Administrator

```
[ActiveEvent(Name="Magix.MetaType.ViewActions")]
protected void Magix_MetaType_ViewActions(object sender, ActiveEventArgs e)
```

Level2: Will show a Grid containing all Meta Actions within the system

```
[ActiveEvent(Name="Magix.MetaActions.SearchActions")]
protected void Magix_MetaActions_SearchActions(object sender, ActiveEventArgs e)
```

Level2: Will show a 'search for Action and select' type of Grid to the end user so that he can select and append an action into whatever collection wants to contain one

```
[ActiveEvent(Name="Magix.MetaAction.ActionWasSelected")]
protected void Magix_MetaAction_ActionWasSelected(object sender, ActiveEventArgs e)
```

Level2: Raise 'ParentPropertyName', first setting the ActionName to the Action found in 'ID'. Helper method for editing Actions in Views

```
[ActiveEvent(Name="DBAdmin.Data.GetContentsOfClass-Filter-Override")]
protected void DBAdmin_Data_GetContentsOfClass_Filter_Override(object sender,
ActiveEventArgs e)
```

Level2: Basically just overrides 'DBAdmin.Data.GetContentsOfClass' to allow for adding some custom Criterias for our Search Box. Puts 'Filter' into a Like expression on the Name column before calling 'base class'

```
[ActiveEvent(Name="Magix.MetaAction.GetCopyActionTemplateColumn")]
protected void Magix_MetaAction_GetCopyActionTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Returns a LinkButton that will allow for Deep-Copying the selected Action

```
[ActiveEvent(Name="Magix.MetaAction.CopyAction")]
protected void Magix_MetaAction_CopyAction(object sender, ActiveEventArgs e)
```

Level2: Performs a Deep-Copy of the Action and returns the ID of the new Action as 'NewID'

```
[ActiveEvent(Name="Magix.MetaAction.CopyActionAndEdit")]
protected void Magix_MetaAction_CopyActionAndEdit(object sender, ActiveEventArgs e)
```

Level2: Performs a Deep-Copy of the Action and start editing the Action immediately

```
[ActiveEvent(Name="Magix.Meta.CreateAction")]
protected void Magix_Meta_CreateAction(object sender, ActiveEventArgs e)
```

Level2: Creates a new Default Action and returns the ID of the new Action as 'NewID'

```
[ActiveEvent(Name="Magix.Meta.CreateActionAndEdit")]
protected void Magix_Meta_CreateActionAndEdit(object sender, ActiveEventArgs e)
```

Level2: Creates a new Default Action and starts editing it immediately

```
[ActiveEvent(Name="Magix.Meta.EditAction")]
protected void Magix_Meta_EditAction(object sender, ActiveEventArgs e)
```

Level2: Edits the given Action ['ID'], with all its properties, parameters and so on. Also creates a 'Run' button which the end user can click to run the action

```
[ActiveEvent(Name="Magix.Meta.DeleteParameter")]
protected void Magix_Meta_DeleteParameter(object sender, ActiveEventArgs e)
```

Level2: Deletes the Action.ActionParams given ['ID'] and updates a lot of UI properties. Raises 'Magix.Core.GetSelectedTreeItem' to get the ID of which ActionParams to actually delete

```
[ActiveEvent(Name="Magix.Meta.CreateParameter")]
protected void Magix_Meta_CreateParameter(object sender, ActiveEventArgs e)
```

Level2: Creates a new Parameter and attaches to either the Selected Parameter, or if none selected, the Action directly on root level. Depends upon the 'Magix.Core.GetSelectedTreeItem' event to get to know whether or not it should add the action to a specific ActionParam as a Child or directly upon the root level of the Action given through 'ID'

```
[ActiveEvent(Name="Magix.MetaAction.DeleteMetaAction")]
protected void Magix_MetaAction_DeleteMetaAction(object sender, ActiveEventArgs e)
```

Level2: Will ask the user for confirmation about deleting the given Action ['ID'], and if the end user confirms, the Action will be deleted

```
[ActiveEvent(Name="Magix.MetaAction.DeleteMetaAction-Confirmed")]
protected void Magix_MetaAction_DeleteMetaAction_Confirmed(object sender, ActiveEventArgs
e)
```

Level2: Will call 'DBAdmin.Common.ComplexInstanceDeletedConfirmed' which again [hopefully] will delete the given Action

```
[ActiveEvent(Name="Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Returns menu items for dashboard functionality to be able to click and view Actions from Dashboard

```
[ActiveEvent(Name="Magix.Core.EventClickedWhileDebugging")]
protected void Magix_Core_EventClickedWhileDebugging(object sender, ActiveEventArgs e)
```

Level2: Will take an incoming 'EventName' with an optionally attached 'EventNode' structure and create an Action out of it

```
[ActiveEvent(Name="Magix.MetaAction.RaiseAction")]
protected void Magix_Meta_RaiseEvent(object sender, ActiveEventArgs e)
```

Level2: Will take an incoming Action ['ActionID' OR 'ActionName'] and run it. Will merge the incoming parameters with the Params of the Action, giving the 'incoming Parameters' preference over the Params associated with Action

```
[ActiveEvent(Name="Magix.MetaAction.EditParam")]
private void Magix_Meta_EditParam(object sender, ActiveEventArgs e)
```

Level2: Will initiate editing of Parameter for Action unless it's already being edited, at which point it'll be 'brought to front'

```
[ActiveEvent(Name="Magix.MetaAction.GetMetaActionParameterTypeNameTemplateColumn")]
private void Magix_MetaAction_GetMetaActionParameterTypeNameTemplateColumn(object
sender, ActiveEventArgs e)
```

Level3: Returns a SelectList with the opportunity for the end user to select which type [system-type, native] the specific parameter should be converted to before Action is being ran

```
[ActiveEvent(Name="Magix.MetaAction.GetActionItemTree")]
private void Magix_Meta_GetActionItemTree(object sender, ActiveEventArgs e)
```

Level2: Returns a tree structure containing all the Action's Parameters to the caller

15. MetaObject_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.MetaObject_Controller

Description:
Level2: Contains logic for editing, maintaining and viewing MetaObjects. MetaObjects are at the heart of the Meta Application System since they serve as the 'storage' for everything a view updates or creates through interaction with the end user

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, ActiveEventArgs e)
```

Level2: Returns menu event handlers for viewing MetaObjects

```
[ActiveEvent(Name="Magix.MetaType.EditMetaObjects_UnFiltered")]
protected void Magix_MetaType_EditMetaObjects_UnFiltered(object sender, ActiveEventArgs e)
```

Level2: Will open up editing of the MetaObject directly, without any 'views' or other interferences. Mostly meant for administrators to edit objects in 'raw mode'

```
[ActiveEvent(Name="Magix.MetaType.GetCopyMetaObjectTemplateColumn")]
protected void Magix_MetaType_GetCopyMetaObjectTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return a 'Copy Template LinkButton' back to caller

```
[ActiveEvent(Name="Magix.MetaType.CreateMetaObject")]
protected void Magix_MetaType_CreateMetaObject(object sender, ActiveEventArgs e)
```

Level2: Creates a new MetaObject with some default values and returns the ID of the new MetaObject as 'NewID'

```
[ActiveEvent(Name="Magix.MetaType.CreateMetaObjectAndEdit")]
protected void Magix_MetaType_CreateMetaObjectAndEdit(object sender, ActiveEventArgs e)
```

Level2: Creates a new MetaObject with some default values, and lets the end user edit it immediately

```
[ActiveEvent(Name="Magix.MetaType.EditONEMetaObject_UnFiltered")]
protected void Magix_MetaType_EditONEMetaObject_UnFiltered(object sender, ActiveEventArgs e)
```

Level2: Allows for editing the MetaObject directly without any Views filtering out anything

```
[ActiveEvent(Name="Magix.MetaType.AppendChildMetaObjectToMetaObject")]
protected void Magix_MetaType_AppendChildMetaObjectToMetaObject(object sender, ActiveEventArgs e)
```

Level2: Will Append an existing MetaObject [ID] to another existing MetaObject [ParentID] as a child

```
[ActiveEvent(Name="Magix.MetaType.AppendChildMetaObjectToMetaObjectAndEditParent")]
protected void Magix_MetaType_AppendChildMetaObjectToMetaObjectAndEditParent(object sender, ActiveEventArgs e)
```

Level2: Will Append an existing MetaObject [ID] to another existing MetaObject [ParentID] as a child and immediately Edit the Parent MetaObject

```
[ActiveEvent(Name="DBAdmin.Form.AppendObject-OverriddenForVisualReasons")]
protected void DBAdmin_Form_AppendObject_OverriddenForVisualReasons(object sender,
ActiveEventArgs e)
```

Level2: Calls DBAdmin.Form.AppendObject after overriding some 'visual properties' for the Grid system

```
[ActiveEvent(Name="Magix.MetaType.CreateNewMetaObject-Value")]
protected void Magix_MetaType_CreateNewMetaObject_Value(object sender, ActiveEventArgs e)
```

Level2: Creates a new 'Value Row' for our MetaObject ['ID']. Returns the ID of the new Value object as 'NewID'

```
[ActiveEvent(Name="Magix.MetaType.CreateNewMetaObject-Value-AndEdit")]
protected void Magix_MetaType_CreateNewMetaObject_Value_AndEdit(object sender,
ActiveEventArgs e)
```

Level2: Creates a new 'Value Row' for our MetaObject and Edits the MetaObject immediately

```
[ActiveEvent(Name="Magix.MetaType.EditONEMetaObject_UnFiltered-ChildMetaObject")]
protected void Magix_MetaType_EditONEMetaObject_UnFiltered_ChildMetaObject(object sender,
ActiveEventArgs e)
```

Level2: Calls 'Magix.MetaType.EditONEMetaObject_UnFiltered' after changing the Container to display the module within. Allows editging of Child MetaObjects

```
[ActiveEvent(Name="Magix.MetaType.GetMetaObjectChildrenTemplateColumn")]
protected void Magix_MetaType_GetMetaObjectChildrenTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Returns a LinkButton with no Children back to caller upon which clicked will start editing the Children collection of objects within the MetaObject

```
[ActiveEvent(Name="Magix.Publishing.RemoveChildObject")]
protected void Magix_Publishing_RemoveChildObject(object sender, ActiveEventArgs e)
```

Level2: Will remove the Child MetaObject ['ID'] from the Parent MetaObject ['ParentID'] collection of children. Notice the child object will NOT be deleted, only 'unreferenced out of' the parent MetaObject

```
[ActiveEvent(Name="Magix.Publishing.RemoveChildObjectAndEdit")]
protected void Magix_Publishing_RemoveChildObjectAndEdit(object sender, ActiveEventArgs e)
```

Level2: Will remove the Child MetaObject ['ID'] from the Parent MetaObject ['ParentID'] collection of children. Notice the child object will NOT be deleted, only 'unreferenced out of' the parent MetaObject. Will instantly edit the Parent MetaObject.

```
[ActiveEvent(Name="Magix.MetaType.GetMetaObjectValuesTemplateColumn")]
protected void Magix_MetaType_GetMetaObjectValuesTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return a TextAreaEdit, from which the Value of the Value object belonging to the MetaObject can be edited, and a LinkButton, from which the entire object can be deleted, back to caller

```
[ActiveEvent(Name="Magix.MetaType.GetMetaObjectValuesNAMETemplateColumn")]
protected void Magix_MetaType_GetMetaObjectValuesNAMETemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return an InPlaceEdit back to caller, since having Carriage Returns in a Property Name would only serve to be ridiculous

```
[ActiveEvent(Name="Magix.MetaType.DeleteMetaObject")]
protected void Magix_MetaType_DeleteMetaObject(object sender, ActiveEventArgs e)
```

Level2: Will ask the user for confirmation to assure he really wants to delete the specific MetaObject ['ID'], and if the user confirms will delete that object

```
[ActiveEvent(Name="Magix.MetaType.DeleteObjectRaw-Confirmed")]
protected void Magix_MetaType_DeleteObjectRaw_Confirmed(object sender, ActiveEventArgs e)
```

Level2: Implementation of deletion of MetaObject after user has confirmed he really wants to delete it

```
[ActiveEvent(Name="DBAdmin.Common.CreateObjectAsChild")]
protected void DBAdmin_Common_CreateObjectAsChild(object sender, ActiveEventArgs e)
```

Level2: Here only to make sure Grids are updated if we're adding a child MetaObject to another MetaObject

```
[ActiveEvent(Name="DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Clears from content4 and out

```
[ActiveEvent(Name="Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Returns the number of MetaObjects in the system back to caller and the name of the Event needed to show all MetaObjects in the system

```
[ActiveEvent(Name="Magix.MetaType.SetMetaObjectValue")]
protected void Magix_MetaType_SetMetaObjectValue(object sender, ActiveEventArgs e)
```

Level2: Will either update the existing Value or create a new Value with the given 'Name' and make sure exists within the MetaObject ['MetaObjectID']

```
[ActiveEvent(Name="DBAdmin.Data.ChangeSimplePropertyValue")]
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure we can traverse our MetaObjects in META mode [front-web, showing grids and views of Meta Objects]

```
[ActiveEvent(Name="DBAdmin.DynamicType.GetObject")]
protected void DBAdmin_DynamicType_GetObject(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be seen 'front-web'

```
[ActiveEvent(Name="Magix.Meta.EditMetaObject")]
protected void Magix_Meta_EditMetaObject(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be edited 'front-web'

```
[ActiveEvent(Name="Magix.Meta.DeleteMetaObject")]
protected void Magix_Meta_DeleteMetaObject(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be deleted 'front-web'

```
[ActiveEvent(Name="Magix.Meta.DeleteMetaObject-Confirmed")]
protected void Magix_Meta_DeleteMetaObject_Confirmed(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be seen 'front-web'. Confirmation, actual deletion

```
[ActiveEvent(Name="Magix.Meta.ChangeMetaObjectValue")]
protected void Magix_Meta_ChangeMetaObjectValue(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can have their values changed 'front-web'

```
[ActiveEvent(Name="Magix.MetaType.CopyMetaObject")]
private void Magix_MetaType_CopyMetaObject(object sender, ActiveEventArgs e)
```

Level2: Will copy the incoming MetaObject ['ID']

16. CommonActions_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.CommonActions_Controller

Description:
Level2: Contains common end user useful actions which doesn't really belong any particular place, but which can still be immensily useful for 'scripting purposes'. Perceive these as 'plugins' ... ;) - [Or extra candy if you wish]. Often they're 'simplifications' of other more 'hard core' Active Events

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Common.GetUserSetting")]
protected void Magix_Common_GetUserSetting(object sender, ActiveEventArgs e)
```

Level2: Returns the 'Name' setting for the current logged in User. Sets the setting to 'Default' if no existing value is found. Both are mandatory params. Returns new value as 'Value'

```
[ActiveEvent(Name="Magix.Common.SetUserSetting")]
protected void Magix_Common_SetUserSetting(object sender, ActiveEventArgs e)
```

Sets the given setting 'Name' to the given value 'Value' for the currently logged in User

```
[ActiveEvent(Name="Magix.Common.SendEmail")]
protected void Magix_Common_SendEmail(object sender, ActiveEventArgs e)
```

Level2: Simplification of 'Magix.Core.SendEmail', will among other things using the User.Current as sender unless explicitly overridden. Will also unless to email address is given, send email to yourself. Takes these parameters 'Header', 'Body', 'Email' [from email], 'From' [name] 'To' [which can be a list of emails or one email]

```
[ActiveEvent(Name="Magix.Common.ReplaceStringValue")]
protected void Magix_Common_ReplaceStringValue(object sender, ActiveEventArgs e)
```

Level2: Will to a String.Replace on the given 'Source' or 'SourceNode'. Will replace 'OldString' or 'OldStringNode' with 'NewString' or 'NewStringNode' and return the value either in 'Result' or 'ResultNode', direct value [no 'Node' part] always have preference

```
[ActiveEvent(Name="Magix.Common.MultiAction")]
protected void Magix_Common_MultiAction(object sender, ActiveEventArgs e)
```

Level2: Will call 'Magix.MetaAction.RaiseAction' for every single 'ActionName' in the Actions [list] Parameter. Useful for creating complex abstractions, doing multiple tasks at once or 'encapsulating' your entire logic inside one Action

```
[ActiveEvent(Name="Magix.Common.RenameNode")]
protected void Magix_Common_RenameNode(object sender, ActiveEventArgs e)
```

Level2: Will rename the given 'FromName' to 'ToName'. Will throw exception if no 'FromName' exists, or parameters are missing

```
[ActiveEvent(Name="Magix.Common.StripAllParametersExcept")]
protected void Magix_Common_StripAllParametersExcept(object sender, ActiveEventArgs e)
```

Level2: Will strip every single Parameter OUT of the Node structure except the given 'But'. But can be either one single name of an object or a list of nodes containing several names. Useful for shrinking nodes as the grow due to being passed around or being parts of MultiActions or something similar

```
[ActiveEvent(Name="Magix.Common.GetSingleMetaObject")]
protected void Magix_Common_GetSingleMetaObject(object sender, ActiveEventArgs e)
```

Level2: Will return the given MetaObject [MetaObjectID] as a Key/Value pair. Will not traverse Child Objects though. Useful for fetching objects for any one reasons you might have, as long as you know their ID

```
[ActiveEvent(Name="Magix.Common.ReloadOriginalWebPart")]
protected void Magix_Common_ReloadOriginalWebPart(object sender, ActiveEventArgs e)
```

Level2: Will reload the Original WebPart, intended to be, within the 'current WebPart container' on the page. Meaning, if you've allowed the user to 'fuzz around all over the place' till he no longer can remember what originally was within a specific WebPart Container, he can raise this event [somehow], which will 'reload the original content' into the 'current container' [container raising the event]

```
[ActiveEvent(Name="Magix.Common.LoadSignatureForCurrentMetaObject")]
protected void Magix_Common_LoadSignatureForCurrentMetaObject(object sender,
ActiveEventArgs e)
```

Level2: If raised from within a MetaView on a specific MetaObject ['MetaObjectID'], somehow, will show the SignatureModule for that particular MetaObject for its 'ActionSenderName' property. When Signature is done [signing complete] the original content of the Container will be reloaded

```
[ActiveEvent(Name="Magix.MetaView.UnLoadSignature")]
protected void Magix_Signature_UnLoadSignature(object sender, ActiveEventArgs e)
```

Level2: Helper for SignatureModule, to store it correctly upon finishing and saving a new Signature. Will extract the 'Signature' content and store into the 'Name' property of the given 'MetaObjectID' MetaObject and save the MetaObject

```
[ActiveEvent(Name="Magix.Common.SetSessionVariable")]
protected void Magix_Common_SetSessionVariable(object sender, ActiveEventArgs e)
```

Level2: Will set the given Session Variable ['Name'] to the 'Value'. Useful for creating caches of huge things, you need to occur really fast [or something]. Session Variables like these can later be retrieved by its sibling method 'Magix.Common.GetSessionVariable'. Things stored into the Session will be on a per user level [meaning, it'll take a LOT of memory on your server], but it will be very fast to retrieve later. Be Cautious here!

```
[ActiveEvent(Name="Magix.Common.GetSessionVariable")]
protected void Magix_Common_GetSessionVariable(object sender, ActiveEventArgs e)
```

Level2: Will return the given Session Variable ['Name'] to the 'Value' output node. Useful for retrieving caches of huge things, you need to occur really fast [or something]. Session Variables like these can be set by its sibling method 'Magix.Common.SetSessionVariable'. Things stored into the Session will be on a per user level [meaning, it'll take a LOT of memory on your server], but it will be very fast to retrieve later. Be Cautious here!

17. CreateDefaultInitialActions_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.CreateDefaultInitialActions_Controller

Description:
Level2: Contains Application Startup code to create the defaul Actions unless they're already there

Basetypes
Magix.Brix.Loader.ActiveController

Methods

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Will create some default installation Actions for the End User to consume in his own Meta Applications. These are all prefixed with 'Magix.DynamicEvent'

18. MetaView_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaViews.MetaView_Controller

Description:
Level2: Contains helper logic for viewing and maintaing MetaViews, and related subjects. MetaViews are the foundation for the whole viewing parts of the Meta Application system. A MetaView is imperativ for both being able to collect new data and also for viewing existing data. The MetaView defines which parts of the object you can see at any time too, which means you can use it to filter access according to which Grid the user is having access to, for instance. This controller contains logic for editing and maintaining MetaViews, plus also direct usage of MetaViews

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, ActiveEventArgs e)
```

Level2: Will return one item back to caller which hopefully will function as the basis of loading the ViewMetaViews logic

```
[ActiveEvent(Name="Magix.MetaType.ViewMetaViewMultiMode")]
protected void Magix_MetaType_ViewMetaViewMultiMode(object sender, ActiveEventArgs e)
```

Level2: Will show the given MetaView ['MetaViewName'] in MultiView mode. As in, the end user will see a Grid of all MetaObjects of the TypeName of the MetaView

```
[ActiveEvent(Name="Magix.MetaView.ViewMetaViews")]
protected void Magix_MetaView_ViewMetaViews(object sender, ActiveEventArgs e)
```

Level2: Will show a Grid with all the MetaViews to the end user

```
[ActiveEvent(Name="Magix.MetaView.MetaView_Single.GetTemplateColumnSelectView")]
protected void Magix_MetaView_MetaView_Single_GetTemplateColumnSelectView(object
sender, ActiveEventArgs e)
```

Level3: Will return a SelectList with all the MetaViews back to caller

```
[ActiveEvent(Name="Magix.MetaView.MetaView_Multiple.GetTemplateColumnSelectView")]
protected void Magix_MetaView_MetaView_Multiple_GetTemplateColumnSelectView(object
sender, ActiveEventArgs e)
```

Level3: Will return a SelectList with all the MetaViews back to caller

```
[ActiveEvent(Name="Magix.Meta.GetCopyMetaViewTemplateColumn")]
protected void Magix_Meta_GetCopyMetaViewTemplateColumn(object sender, ActiveEventArgs
e)
```

Level3: Creates a Copy MetaView LinkButton

```
[ActiveEvent(Name="Magix.Meta.CopyMetaView")]
protected void Magix_Meta_CopyMetaView(object sender, ActiveEventArgs e)
```

Level2: Will copy [deep clone] the incoming 'ID' MetaView and return the new copy's ID as 'NewID'

```
[ActiveEvent(Name="Magix.Meta.CopyMetaViewAndEditCopy")]
protected void Magix_Meta_CopyMetaViewAndEditCopy(object sender, ActiveEventArgs e)
```

Level2: Will copy [deep clone] the incoming 'ID' MetaView and edit it immediately

```
[ActiveEvent(Name="Magix.MetaView.DeleteMetaView")]
protected void Magix_MetaView_DeleteMetaView(object sender, ActiveEventArgs e)
```

Level2: Will ask the user if he really wish to delete the MetaView given through 'ID', and if the user says yes, delete it

```
[ActiveEvent(Name="Magix.MetaView.DeleteMetaView-Confirmed")]
protected void Magix_MetaView_DeleteMetaView_Confirmed(object sender, ActiveEventArgs e)
```

Level2: End user confirmed he wishes to delete the MetaView

```
[ActiveEvent(Name="Magix.MetaView.CreateMetaView")]
protected void Magix_MetaView_CreateMetaView(object sender, ActiveEventArgs e)
```

Level2: Will create a new MetaView with some default settings and return ID of new view as 'NewID'

```
[ActiveEvent(Name="Magix.MetaView.CreateMetaViewAndEdit")]
protected void Magix_MetaView_CreateMetaViewAndEdit(object sender, ActiveEventArgs e)
```

Level2: Will create and Edit immediately a new MetaView

```
[ActiveEvent(Name="Magix.MetaView.EditMetaView")]
protected void Magix_MetaView_EditMetaView(object sender, ActiveEventArgs e)
```

Level2: Will edit the given ['ID'] MetaView with its properties and some other controls. Such as for instance a 'View WYSIWYG' button

```
[ActiveEvent(Name="Magix.MetaView.GetMetaViewActionTemplateColumn")]
protected void Magix_MetaView_GetMetaViewActionTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return a clickable and drop-down-able Panel that the end user can click to append and remove events that triggers the MetaView Property if clicked

```
[ActiveEvent(Name="Magix.MetaView.AppendAction")]
protected void Magix_MetaView_AppendAction(object sender, ActiveEventArgs e)
```

Level2: Will show a Search box, from which the end user can search for a specific action to append to the list of actions already associated with the MetaView Property

```
[ActiveEvent(Name="Magix.MetaView.ActionWasChosenForAppending")]
protected void Magix_MetaView_ActionWasChosenForAppending(object sender, ActiveEventArgs e)
```

Level2: Will append the specific 'ActionName' into the 'ParentID' MetaView Property and call for an update of the grids

```
[ActiveEvent(Name="DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure we clear content4 and out if a MetaView s deleted.

```
[ActiveEvent(Name="Magix.MetaType.ShowMetaViewMultipleInCurrentContainer")]
protected void Magix_MetaType_ShowMetaViewMultipleInCurrentContainer(object sender, ActiveEventArgs e)
```

Level2: Will show the 'MetaViewName' MetaView within the 'current container'

```
[ActiveEvent(Name="DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, ActiveEventArgs e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode', front-web style

```
[ActiveEvent(Name="DBAdmin.DynamicType.CreateObject")]
protected void DBAdmin_DynamicType_CreateObject(object sender, ActiveEventArgs e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode'

```
[ActiveEvent(Name="DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, ActiveEventArgs e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode'

```
[ActiveEvent(Name="Magix.MetaView.MultiViewActionItemTemplateColumn")]
protected void Magix_MetaView_MultiViewActionItemTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return a Button control to caller, upon which clicked, will raise the named actions in its settings according to which MetaViewProperty it belongs to. Front-web stuff, basically a MetaView form field with 'Actions' associated with it

```
[ActiveEvent(Name="Magix.MetaView.RunActionsForMetaViewProperty")]
protected void Magix_MetaView_RunActionsForMetaViewProperty(object sender,
ActiveEventArgs e)
```

Level2: Will run the Actions associated with the MetaViewProperty given through 'MetaViewName' [MetaView - Name], 'Name' [of property within MetaObject] and expects to be raised from within a WebPart, since it will pass along the 'current container' onwards

```
[ActiveEvent(Name="Magix.MetaView.MetaView_Multiple_GetColonTemplateColumn")]
protected void Magix_MetaView_MetaView_Multiple_GetColonTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will create a control type depending upon the colon-prefix of the column. For instance, if given date:When it will create a Calendar, putting the Value selected into the 'When' property. If given select:xx.yy:zz it will create a select list, enumerating into the ObjectTypes given with the Property de-referenced. E.g. select:Gender.Sex:Male-Female will enumerate every 'Sex' value of every object of TypeName 'Gender' and put it into the property with the name of 'Male-Female'. choice:Gender.Sex,Css:Male-Female would function identically, except it also expects to find a Css property, whos value will be used as the CSS class for a small 'clickable enumerating' type of control which would allow for 'single-choice' selection of status for instance. If you want to create plugin types for the MultiView Grid system, this event is what you'd have to handle to inject your own Column types

```
[ActiveEvent(Name="Magix.MetaView.LoadWysiwyg")]
protected void Magix_MetaView_LoadWysiwyg(object sender, ActiveEventArgs e)
```

Level2: Loads up WYSIWYG editor for MetaView in 'SingleView Mode'

```
[ActiveEvent(Name="Magix.MetaView.GetViewData")]
protected void Magix_MetaView_GetViewData(object sender, ActiveEventArgs e)
```

Level2: Returns the properties for the MetaView back to caller

```
[ActiveEvent(Name="Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Returns the number of MetaViews and an event for viewing all MetaViews back to caller

```
[ActiveEvent(Name="Magix.MetaView.CreateSingleViewMetaObject")]
protected void Magix_MetaView_CreateSingleViewMetaObject(object sender, ActiveEventArgs e)
```

Level2: Will create a new MetaObject according to the values given from the MetaView_SingleView form. 'PropertyValues' is expected to contain a Name/Value list-pair, and the ViewName is supposed to be the unique name to the specific MetaView being used.

19. PDF_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.PDF.PDF_Controller

Description:
Level2: Contains logic for creating PDF files and dowloading to Client or saving locally

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.PDF.CreatePDF-Book")]
protected void Magix_PDF_CreatePDF_Book(object sender, ActiveEventArgs e)
```

Level2: Will create a PDF document and redirect the user to that document as named in the 'File' parameter. Takes many different parameters; 'FrontPage' [optional] being an image [600x846 px big] to a cover page you'd like to use. 'Index' [optional] containing all components needed to build the books 'index'. 'Chapters' contains a list of all the pages, with HTML parsing capabilities [one level only]

20. Logging_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Logging_Controller

Description:
Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.ViewLog")]
private void Magix_Publishing_ViewLog(object sender, ActiveEventArgs e)
```

Level2: Will display the Grid of the latest Log Items for the user to be able to drill down, and have a more thourough look

```
[ActiveEvent(Name="Magix.Publishing.GetLogItems")]
private void Magix_Publishing_GetLogItems(object sender, ActiveEventArgs e)
```

Level2: Will return the last log items according to newest first

```
[ActiveEvent(Name="Magix.Publishing.ViewLogItem")]
protected void Magix_Publishing_ViewLogItem(object sender, ActiveEventArgs e)
```

Level2: Will show one LogItem in a grid for the end-user to scrutinize. Will allow the user to always have to LogItems up at the same time

21. LoginOut_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.LoginOut_Controller

Description:
Level2: Login and Logout Controller

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Core.LogInUser")]
protected void Magix_Core_LogInUser(object sender, ActiveEventArgs e)
```

Level2: Basically just checks to see if the given username/password combination exists, and if so sets the User.Current object, which is static per request, returning the logged in user. Then if successful, raises the 'Magix.Core.UserLoggedIn' event

```
[ActiveEvent(Name="Magix.Core.UserLoggedIn")]
protected void Magix_Core_UserLoggedIn(object sender, ActiveEventArgs e)
```

Level2: Redirects the user to Root, unless another Redirect path is somehow given

```
[ActiveEvent(Name="Magix.Publishing.GetStateForLoginControl")]
protected void Magix_Publishing_GetStateForLoginControl(object sender, ActiveEventArgs e)
```

Level2: Returns whether or not the User is logged in or not for the LoginInOut Module to know what types of controls to load

```
[ActiveEvent(Name="Magix.Core.UserLoggedOut")]
private void Magix_Core_UserLoggedOut(object sender, ActiveEventArgs e)
```

Level2: Resets the User.Current object, and redirects user back to root

22. FileSystem_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.FileSystem_Controller

Description:
Level2: Tiny helper for menu item in Administrator Dashboard to view the File System Browser

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.ViewFileSystem")]
protected void Magix_Publishing_ViewFileSystem(object sender, ActiveEventArgs e)
```

Level2: Will call 'Magix.FileExplorer.LaunchExplorer' with a couple of predefined values for positioning and such. Clears all containers from 4 and out

23. InitialLoading_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.InitialLoading_Controller

Description:
Level2: Class for taking care of the Magix.Core.InitialLoading message, meaning the initialization of the page on a per User level initially as they're coming to a new 'page' or URL ... Basically just handles Magix.Core.InitialLoading and mostly delegates from that point ...

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Core.InitialLoading")]
protected void Magix_Core_InitialLoading(object sender, ActiveEventArgs e)
```

Level2: This method will handle the 'initial loading', meaning basically that the page was loaded initially by either changing the URL of the browser to the app or doing a Refresh or something. For coders, meaning basically not IsPostBack on the page object ... Throws a whole range of different events based upon whether or not the User is logged in or not, and which URL is being specifically requested. Its most noticable outgoing event would though be 'Magix.Publishing.LoadDashboard' and 'Magix.Publishing.UrlRequested'. PS! If you override this one, you've effectively completely bypassed every single existing logic in Magix, and everything are 'dead event handlers' in 'limbo' not tied together at all. Which might be cool, or might be a nightmare, depending upon how you use it, if you use it, use it with CARE if you do though !!

24. SliderMenu_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.SliderMenu_Controller

Description:
Level2: Helps feed the SliderMenu in Front-Web to get its Items to build its structure upon

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetSliderMenuItems")]
protected void Magix_Publishing_GetSliderMenuItems(object sender, ActiveEventArgs e)
```

Level2: Will return a node containing all the menu items in your Pages hierarchy, possibly according to how you've got access to them, as long as the Access Controller is in no ways jeopardized

```
[ActiveEvent(Name="Magix.Publishing.SliderMenuItemClicked")]
protected void Magix_Publishing_SliderMenuItemClicked(object sender, ActiveEventArgs e)
```

Level2: Will Open the specific WebPage accordingly to which SlidingMenuItem was clicked

```
private void GetOneMenuItem(Node node, WebPage po, bool isRoot)
```

Level2: Will build up the Node structure for one Menu item, first verifying the User has access to that particular page by raising the 'Magix.Publishing.GetRolesListForPage' event

25. TipOfToday_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.TipOfToday_Controller

Description:
Level2: Creats our default Tooltips or Tutorial rather

Basetypes
Magix.Brix.Loader.ActiveController

Methods

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Creates our default tooltips if there exists none

26. OpenID_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.OpenID_Controller

Description:
Level2: Controller implementing the OpenID logic for both Relying Party and OpenID Provider logic. Every User in Magix can use the website where Magix is installed as an OpenID provider by typing in the root URL of the web application and append ?openID=username In addition you can associate as many OpenID tokens as you wish to your User and then use these other OpenID providers to log into Magix.

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)
```

Level2: Overriding our default Magix Login logic here to inject our own OpenID stuff ... Overrides; 'Magix.Core.LogInUser' and 'Magix.Core.UserLoggedIn' which again are being used by the Magix 'core' login logic, meaning we're basically doing effectively 'AOP' here by overriding existing events like these ... Where the 'Aspect' here would be the OpenID 'logic'

```
[ActiveEvent(Name="Magix.Core.InitialLoading")]
protected void Magix_Core_InitialLoading(object sender, ActiveEventArgs e)
```

Level2: Basically just checking if this is an OpenID Request, and doing either the Provider or the Relying Party thing needed ...

```
[ActiveEvent(Name="Magix.Publishing.NewUserRegisteredThroughOpenID")]
protected void Magix_Publishing_NewUserRegisteredThroughOpenID(object sender,
ActiveEventArgs e)
```

Level2: Creates a default User object, and ads him into the default role as set through the 'Magix.Publishing.OpenID.DefaultRoleName' setting. Associates the given OpenID Token with the User [obviously] Will also try to extract additional information, such as Address etc from the OpenID Provider ...

```
[ActiveEvent(Name="Magix.Core.LogInUser-Override")]
protected void Magix_Core_LogInUser_Override(object sender, ActiveEventArgs e)
```

Level2: Overridden to check if we're in OpenID Mode or in Username/Password mode Will forward to overridden method is no OpenID value is given ...

```
[ActiveEvent(Name="Magix.Core.UserLoggedIn-Override")]
protected void Magix_Core_UserLoggedIn_Override(object sender, ActiveEventArgs e)
```

Level2: Overridden to check to see if we're an OpenID Provider, and if so, redirect back to Relying Party accordingly. If we're not a Provider, it will forward the event to the overridden logic

```
private void InjectOpenIDHeaderHTMLMetaElements()
```

Helper method used to inject OpenID Meta information into the header element of the Page.

```
private void ProcessAnyOpenIDProviderRequests()
```

Will try to extract DotNetOpenAuth OpenID requests being passed around, and process them, if any

```
private void ProcessAnyOpenIDRelyingPartyRequests()
```

Will check to see if there's any OpenID Relying Party requests currently coming in, or in the pipeline, and process them if necessary.

```
private void OpenIDProviderVerifiedToken(IAuthenticationResponse r)
```

OpenID Token was verified, can we log in directly, or are we supposed to create a new User object. As in is this OpenID Token associated with an existing User here ...?

```
private bool AuthenticateUserFromIncomingOpenID(IAuthenticationResponse r)
```

Will either log in existing user from an OpenID Token and return true, or not be able to match him with an existing user and hence return false

```
private void LogInExistingUserFromOpenIDToken(IAuthenticationResponse r, OpenIDToken
token)
```

Will log in User from an existing OpenID Token, and show a message box back to the end user to show him we had success

```
private void CreateNewUserFromOpenIDToken(IAuthenticationResponse r)
```

Will raise the 'Magix.Publishing.NewUserRegisteredThroughOpenID' with all the data given from the OpenID Provider. And show the end user a dialog confirming he's successfully authenticated

27. PageLoader_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.PageLoader_Controller

Description:
Level2: Controller responsible for loading up 'Page Objects', and doing initialization around pages and such as they're being loaded

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.UrlRequested")]
protected void Magix_Publishing_UrlRequested(object sender, ActiveEventArgs e)
```

Level2: Basically just 're-maps' the event to Magix.Publishing.OpenPage and changes the incoming URL parameter to an outgoing ID parameter of the specific page being requested ...

```
[ActiveEvent(Name="Magix.Publishing.OpenPage")]
protected void Magix_Publishing_OpenPage(object sender, ActiveEventArgs e)
```

Level2: Expects an ID parameter which should be the ID of a Page. Will loop through every WebPart of the page, and raise 'Magix.Publishing.InjectPlugin' for every WebPart within the WebPage object. Might throw exceptions if the page is not found, or the user has no access rights to that specific page or something ... Will verify the user has access to the page, by raising 'Magix.Publishing.VerifyUserHasAccessToPage' and if no access is granted, try to find the first child object the user has access to through raising the 'Magix.Publishing.FindFirstChildPageUserCanAccess' event. Will throw SecurityException or ArgumentException if page not found or granted access to [nor any of its children]

```
[ActiveEvent(Name="Magix.Core.GetContainerForControl")]
protected void Magix_Core_GetContainerForControl(object sender, ActiveEventArgs e)
```

Level2: Will return the Container ID of the 'Current Container', meaning whomever raised whatever event we're currently within. Meaning if you've got a Button in a Grid which raises some event which is dependent upon knowing which Container its being raised from within, to load some other control [e.g. Signature Column in Grid or Edit Object] Is dependent upon that the PageObjectTemplateID is being passed around some way or another into this bugger

```
private void IncludeStaticFrontEndCSS()
```

Includes static CSS for our front-end

```
private void OpenPage(WebPage page)
```

Helper method used in 'Magix.Publishing.OpenPage' for opening the page once it has been granted access to either directly, or one of its child pages. Will basically just loop through every WebPart in the Page and raise 'Magix.Publishing.InjectPlugin' for each of them, after doing some manipulation of the Container the WebPart is suppsed to become injected into and such. Will also set the title of the page according to the name of the WebPage

```
private void SetCaptionOfPage(WebPage page)
```

Sets the title of the page by raising 'Magix.Core.SetTitleOfPage' event according to the Name of the Page

28. WebPart_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.WebPart_Controller

Description:
Level2: Initializes WebParts and such while being injected onto WebPage

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.InitializePublishingPlugin")]
protected void Magix_Publishing_InitializePublishingPlugin(object sender, ActiveEventArgs e)
```

Level2: Initializes the Plugin with its values according to what's in its settings

```
[ActiveEvent(Name="Magix.Publishing.ReloadWebPart")]
protected void Magix_Publishing_ReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Reloads the Original WebPart content

```
[ActiveEvent(Name="Magix.Publishing.GetWebPartSettingValue")]
private void Magix_Publishing_GetWebPartSettingValue(object sender, ActiveEventArgs e)
```

Level2: Returns the Value of a specific setting of a specific WebPart

```
[ActiveEvent(Name="Magix.Publishing.InjectPlugin")]
private void Magix_Publishing_InjectPlugin(object sender, ActiveEventArgs e)
```

Level2: Injects one plugin into the given container, unless anything says the Plugin doesn't need to be reloaded for some reasons [sliding menu for instance]

29. AdministratorDashboard_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.AdministratorDashboard_Controller

Description:
Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.LoadAdministratorDashboard")]
protected void Magix_Publishing_LoadAdministratorDashboard(object sender, ActiveEventArgs e)
```

Level2: Loads Administrator Dashboard [back-web]

```
[ActiveEvent(Name="Magix.Publishing.LoadAdministratorMenu")]
protected void Magix_Publishing_LoadAdministratorMenu(object sender, ActiveEventArgs e)
```

Level2: Loads the Administrator SlidingMenu into the 1st content, but everything here is basically overridable. Will also raise 'Magix.Publishing.GetPluginMenuItems' to allow for plugins to connect up their own Administrator Dashboard menu items

```
[ActiveEvent(Name="Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Ads up the 'common data' for the Admin Dashboard such as Users, Roles etc

```
[ActiveEvent(Name="Magix.Publishing.ViewClasses")]
private void Magix_Publishing_ViewClasses(object sender, ActiveEventArgs e)
```

Level2: Will fire up our Database Manager

30. ChildExcerpt_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.ChildExcerpt_Controller

Description:
Level2: Helps out with some of the editing and using of the ChildExcerpt Module Type

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.GetEditChildExcerptNoArticlesDropDown")]
private void Magix_Publishing_GetEditChildExcerptNoArticlesDropDown(object sender,
ActiveEventArgs e)
```

Level3: Creates a SelectList from which the no of articles in the ChildExcerpt Object can be chosen

```
[ActiveEvent(Name="Magix.Publishing.BuildOneChildExcerptControl")]
private void Magix_Publishing_BuildOneChildExcerptControl(object sender, ActiveEventArgs e)
```

Level3: The actual construction of our Child Excerpt panels are being done here

```
[ActiveEvent(Name="Magix.Publishing.GetLastChildrenPages")]
private void Magix_Publishing_GetLastChildrenPages(object sender, ActiveEventArgs e)
```

Level2: Will return the last 'Count' number of pages, sorted according to newest first

```
[ActiveEvent(Name="Magix.Publishing.GetTemplateColumnSelectChildExcerptNo")]
protected void Magix_Publishing_GetTemplateColumnSelectChildExcerptNo(object sender,
ActiveEventArgs e)
```

Level3: Will create a SelectList for editing of the number of ChildExcerpts in the WebPart

31. EditUser_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditUser_Controller

Description:
Level2: Here mostly for Editing User, and also to some extent creating default users and roles upon startup if none exists

Basetypes
Magix.Brix.Loader.ActiveController

Methods

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Makes sure we have Administrator and User Roles, in addition creates a default user [admin/admin] if no user exists

[ActiveEvent(Name="Magix.Publishing.EditUsers")]
protected void Magix_Publishing_EditUsers(object sender, ActiveEventArgs e)

Level2: Allows editing of all users, and searching and changing properties and such. Shows all Users in a Grid

[ActiveEvent(Name="Magix.Publishing.EditUser")]
protected void Magix_Publishing_EditUser(object sender, ActiveEventArgs e)

Level2: Edit one specific User, and all of his properties

```
[ActiveEvent(Name="Magix.Publishing.CreateUser")]
protected void Magix_Publishing_CreateUser(object sender, ActiveEventArgs e)
```

Level2: Callback for creating a New User from Dashboard that pops up editing the specific user after his initial creation

```
[ActiveEvent(Name="Magix.Publishing.ChangeAvatarForUser")]
protected void Magix_Publishing_ChangeAvatarForUser(object sender, ActiveEventArgs e)
```

Level2: User has requested a change of Avatar for either himself, or another user. Loads up the FileExplorer Selector, and allows for selecting a new avatar for the admin user

```
[ActiveEvent(Name="Magix.Publishing.SelectImageForUser")]
protected void Magix_Publishing_SelectImageForUser(object sender, ActiveEventArgs e)
```

Level2: New image was selected in our FileExplorer as a new Avatar for our User. Will change the User's Avatar and save it and update

```
[ActiveEvent(Name="Magix.Publishing.GetDefaultAvatarURL")]
protected void Magix_Publishing_GetDefaultAvatarURL(object sender, ActiveEventArgs e)
```

Level2: Callback for supplying the Default Avatar for a User. Override this one if you wish to change the default image for a new user, which might be useful in e.g. OpenID scenarios and profiling efforts

```
[ActiveEvent(Name="Magix.Publishing.GetRoleTemplateColumn")]
private void Magix_Publishing_GetRoleTemplateColumn(object sender, ActiveEventArgs e)
```

Level3: Get 'select Role' template column for our Grid. Creates a SelectList and returns back to caller. PS! We only allow for editing the user such that he or she can belong to only one Role. If you'd like to extend this logic to allow for more than one Role to be added, this is where you'd want to do such, probably by overriding this event, and choose some sort of 'multiple choice' control to return instead

32. AdminDashboardHeader_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.AdminDashboardHeader_Controller

Description:
Level2: We'll 'lock' the Header control since it can be VERY annoying sometimes due to the DB Manager, which seriously needs to be refactored here BTW ...

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.LoadHeader")]
protected void Magix_Publishing_LoadHeader(object sender, ActiveEventArgs e)
```

Level2: Basically just ensures the Header Menu is loaded, and locked with 'Administrator Dashboard' as its value.

33. EditRoles_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditRoles_Controller

Description:
Level2: Here mostly to serve up grids and such for editing of Roles in the system

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.EditRoles")]
protected void Magix_Publishing_EditRoles(object sender, ActiveEventArgs e)
```

Level2: Will allow the end-user to edit [add, change, delete] roles within the system

34. EditTemplates_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditTemplates_Controller

Description:
Level2: Controller for editing Templates. Contains all the relevant event handlers and logic for editing your templates

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.EditTemplates")]
protected void Magix_Publishing_EditTemplates(object sender, ActiveEventArgs e)
```

Level2: Shows the grid containing all your Templates such that you can edit them or create new Templates and such

```
[ActiveEvent(Name="DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Only here to make sure that if a WebPageTemplate is being deleted from the Grid while editing it [for instance], we clear from 'content4' and out to make sure we're not allowig for editing of a deleted Template

```
[ActiveEvent(Name="Magix.Publishing.GetCssTemplatesForWebPartTemplate")]
protected void Magix_Publishing_GetCssTemplatesForWebPartTemplate(object sender,
ActiveEventArgs e)
```

Level2: Will return all possible 'CSS Template Values' from the 'web-part-templates.css' file within the media/module/ folders

```
[ActiveEvent(Name="Magix.Publishing.ChangeCssForWebPartTemplateFromCssTemplate")]
protected void Magix_Publishing_ChangeCssForWebPartTemplateFromCssTemplate(object
sender, ActiveEventArgs e)
```

Level2: Changes the CSS class of the WebPartTemplate instance according to the Selected Css value chosen

```
[ActiveEvent(Name="Magix.Publishing.CopyTemplate")]
protected void Magix_Publishing_CopyTemplate(object sender, ActiveEventArgs e)
```

Level2: Completes a 'deep copy' of the Template and saves it and loads up the Editing of it for the user to immediately start editing it

```
[ActiveEvent(Name="Magix.Publishing.EditTemplate")]
protected void Magix_Publishing_EditTemplate(object sender, ActiveEventArgs e)
```

Level2: Edits one specific WebPageTemplate by loading up the 'Magix.Brix.Components.ActiveModules.Publishing.EditSpecificTemplate' module

```
[ActiveEvent(Name="Magix.Publishing.DeleteWebPartTemplate")]
protected void Magix_Publishing_DeleteWebPartTemplate(object sender, ActiveEventArgs e)
```

Level2: Deletes a specific WebPartTemplate

```
[ActiveEvent(Name="Magix.Publishing.GetWebPageTemplates")]
protected void Magix_Publishing_GetWebPageTemplates(object sender, ActiveEventArgs e)
```

Level2: Returns a node of all WebPageTemplates in the system

```
[ActiveEvent(Name="Magix.Publishing.GetPublisherPlugins")]
protected void Magix_Publishing_GetPublisherPlugins(object sender, ActiveEventArgs e)
```

Level2: Returns a node of all PublisherPlugins you've got available in your installation

```
[ActiveEvent(Name="Magix.Publishing.ChangeModuleTypeForWebPartTemplate")]
protected void Magix_Publishing_ChangeModuleTypeForWebPartTemplate(object sender,
ActiveEventArgs e)
```

Level2: Changes the ModuleName [plugin type] of the WebPartTemplate and saves it

```
[ActiveEvent(Name="Magix.Publishing.ChangeTemplateProperty")]
protected void Magix_Publishing_ChangeTemplateProperty(object sender, ActiveEventArgs e)
```

Level2: Changes properties of the Template and saves it

```
[ActiveEvent(Name="Magix.Publisher.GetNoContainersTemplateColumn")]
protected void Magix_Publisher_GetNoContainersTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Creates the Container column [select list] for selecting different number of WebParts for your template

```
[ActiveEvent(Name="Magix.Publisher.ChangeNumberOFContainersOnTemplates-Confirmed")]
protected void Magix_Publisher_ChangeNumberOFContainersOnTemplates_Confirmed(object
sender, ActiveEventArgs e)
```

Level2: Will change the number of WebParts in your WebPartTemplate

```
[ActiveEvent(Name="Magix.Publisher.GetCopyTemplateColumn")]
private void Magix_Publisher_GetCopyTemplateColumn(object sender, ActiveEventArgs e)
```

Level3: Creates the 'Copy Template' LinkButtons and returns back to the Grid system so that we can have an implementation of 'Copy Template' button. LinkButton raises 'Magix.Publishing.CopyTemplate' when clicked

```
private void RaiseTryToChangeNumberOfWebParts(WebPageTemplate t, int count)
```

Level2: Checks to see if number of webparts are decreasing, if they are, warning user about that he should be careful, blah, blah, blah ...

35. Dashboard_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Dashboard_Controller

Description:
Level2: Main 'router' in dispatching important Dashboard functionality

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.LoadDashboard")]
protected void Magix_Publishing_LoadDashboard(object sender, ActiveEventArgs e)
```

Level2: Will load the Dashboard if User is of type Administrator through raising 'Magix.Publishing.LoadDashboard', otherwise raise 'Magix.Publishing.LoadUserDashboard' which isn't currently being catched any place

36. EditPages_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditPages_Controller

Description:
Level2: Contains logic for Editing WebPage objects and such for administrator

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)
```

Level2: Overridden to make sure we've got some default pages during startup if none exists from before

```
[ActiveEvent(Name="Magix.Publishing.EditPages")]
protected void Magix_Publishing_EditPages(object sender, ActiveEventArgs e)
```

Level2: Will load a Tree of the Pages from which you can browse around within to edit and view the relationships between all your WebPage objects

```
[ActiveEvent(Name="Magix.Publishing.GetEditPagesDataSource")]
protected void Magix_Publishing_GetEditPagesDataSource(object sender, ActiveEventArgs e)
```

Level2: Will return the WebPages in your system back to caller in the return value as 'Items' collections

```
[ActiveEvent(Name="Magix.Publishing.EditSpecificPage")]
protected void Magix_Publishing_EditSpecificPage(object sender, ActiveEventArgs e)
```

Level2: Will edit one Specific page according to either SelectedItemID [Tree] or ID parameter

```
[ActiveEvent(Name="Magix.Publishing.PageWasUpdated")]
protected void Magix_Publishing_PageWasUpdated(object sender, ActiveEventArgs e)
```

Level2: Will make sure we update our Tree control

```
[ActiveEvent(Name="Magix.Publishing.ChangePageProperty")]
protected void Magix_Publishing_ChangePageProperty(object sender, ActiveEventArgs e)
```

Level2: Changes the given properties of the WebPage object. Legal values are Text, URL, PageTemplateID

```
[ActiveEvent(Name="Magix.Publishing.CreateChild")]
protected void Magix_Publishing_CreateChild(object sender, ActiveEventArgs e)
```

Level2: Will create a new WebPage being the child of the given 'ID' WebPage

```
[ActiveEvent(Name="Magix.Publishing.DeletePageObject")]
protected void Magix_Publishing_DeletePageObject(object sender, ActiveEventArgs e)
```

Level2: Will ask the end user if he wish to delete specific Page object and all of its children

```
[ActiveEvent(Name="Magix.Publishing.DeletePageObject-Confirmed")]
protected void Magix_Publishing_DeletePageObject_Confirmed(object sender, ActiveEventArgs
e)
```

Level2: Will delete specific Page object ['ID'] and all of its children

```
[ActiveEvent(Name="Magix.Publishing.ChangeWebPartSetting")]
protected void Magix_Publishing_ChangeWebPartSetting(object sender, ActiveEventArgs e)
```

Level2: Will update and save the incoming WebPartID [WebPartSetting] and set its Value property to 'Value'

```
[ActiveEvent(Name="Magix.Publishing.WebPartTemplateWasModified")]
protected void Magix_Publishing_WebPartTemplateWasModified(object sender, ActiveEventArgs e)
```

Level3: Will run through all WebParts which are 'touched' by this WebPartTemplate and update their settings according to whatever new module was chosen. Warning; This will set any 'value properties' in these WebParts to their default value. It might also retrieve values it had before if this WebPart has earlier been this type of WebPartTemplate

```
[ActiveEvent(Name="Magix.Publishing.WebPageTemplateWasModified")]
protected void Magix_Publishing_WebPageTemplateWasModified(object sender, ActiveEventArgs e)
```

Level2: Will update all WebPages that was modified by changing the WebPageTemplate. This might include creating new WebParts with default values, or removing existing WebParts on Pages

37. Access_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Access_Controller

Description:
Level3: Helps out sorting out which Pages and Menu Items which Users and Roles and such have access to

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Publishing.VerifyUserHasAccessToPage")]
protected void Magix_Publishing_VerifyUserHasAccessToPage(object sender, ActiveEventArgs e)
```

Level3: Returns 'STOP' to true unless User has access to Page, either explicitly or implicitly

```
[ActiveEvent(Name="Magix.Publishing.FindFirstChildPageUserCanAccess")]
protected void Magix_Publishing_FindFirstChildPageUserCanAccess(object sender,
ActiveEventArgs e)
```

Level3: Will recursively traverse children until it find the first page [breadth first] User has access to from given page

```
[ActiveEvent(Name="Magix.Publishing.GetRolesListForPage")]
protected void Magix_Publishing_GetRolesListForPage(object sender, ActiveEventArgs e)
```

Level3: Will return a List of Roles that have explicit access [or not] to the given Page

```
[ActiveEvent(Name="Magix.Publishing.ChangePageAccess")]
protected void Magix_Publishing_ChangePageAccess(object sender, ActiveEventArgs e)
```

Level3: Will change the Access rights for a specific page

```
[ActiveEvent(Name="Magix.Publishing.PageObjectDeleted")]
protected void Magix_Publishing_PageObjectDeleted(object sender, ActiveEventArgs e)
```

Level3: Handled to make sure we delete all WebPageRoleAccess objects belonging to WebPage too

38. Settings_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Settings.Settings_Controller

Description:
Level2: Helper logic to assist with Settings and updating of settings. PS! Unless you have this controller in your Application Pool, Enabled, then direct changes, through the DB Admin module to Settings will NOT affect the Application Pool before it's being re started ...

Methods

```
private void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)
```

Level2: Need to override 'DBAdmin.Data.ChangeSimplePropertyValue' to make sure settings are 'flushed' if they're being updated

```
[ActiveEvent(Name="DBAdmin.Data.ChangeSimplePropertyValue-Override")]
protected void DBAdmin_Data_ChangeSimplePropertyValue_Override(object sender,
ActiveEventArgs e)
```

Level2: Overrides 'DBAdmin.Data.ChangeSimplePropertyValue' to make sure settings are 'flushed' if they're being updated. Calls original logic, then resets the settings and reloads them. This because settings are cached on Application Level for speed issues

39. Signature_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Signature.Signature_Controller

Description:
Level2: Signature Plugin Control for Publishing system to allow for loading of a different plugin module from e.g. the click of a row button or a single-view button etc.

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Signature.LoadSignature")]
protected void Magix_Signature_LoadSignature(object sender, ActiveEventArgs e)
```

Level2: Will load the Signature module, asking for Signature from the user and saving associated with the record. Will replace the module that raised the Action with the Signature module

40. TalkBack_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.TalkBack.TalkBack_Controller

Description:
Level2: Contains helper logic for the TalkBack module [forums] in Magix

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Talkback.GetPostings")]
protected void Magix_Talkback_GetPostings(object sender, ActiveEventArgs e)
```

Level3: Will fetch the top 5 postings with the most recent activity within

```
[ActiveEvent(Name="Magix.TalkBack.LaunchForum")]
protected void Magix_TalkBack_LaunchForum(object sender, ActiveEventArgs e)
```

Level2: Will launch the Talkback Module and allow for posting questions and such

```
[ActiveEvent(Name="Magix.Talkback.CreatePost")]
protected void Magix_Talkback_CreatePost(object sender, ActiveEventArgs e)
```

Level3: Will create a new posting with the Header and Body combination linked to the User.Current. If Parent, will put it into the Parent as a Reply

41. ToolTip_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.ToolTip.ToolTip_Controller

Description:
Level2: Contains helper logic for displaying and maintaining previous/next Tip of today and such

Basetypes
Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name="Magix.Core.GetPreviousToolTip")]
private void Magix_Core_GetPreviousToolTip(object sender, ActiveEventArgs e)
```

Level2: Returns the Previous Tooltip on a 'Per User' level. Meaning, it'll keep track of which ToolTip has been shown to which User

```
[ActiveEvent(Name="Magix.Core.GetNextToolTip")]
private void Magix_Core_GetNextToolTip(object sender, ActiveEventArgs e)
```

Level2: Returns the Next Tooltip on a 'Per User' level. Meaning, it'll keep track of which ToolTip has been shown to which User

```
[ActiveEvent(Name="Magix.Core.GetAllToolTips")]
private void Magix_Core_GetAllToolTips(object sender, ActiveEventArgs e)
```

Level2: Returns all ToolTip instances to caller

42. Separator

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Separator

Description:
Level2: Encapsulates a [dead] html Horizontal Ruler [hr] element for you if you need one

43. TipOfToday

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.TipOfToday

Description:
Level2: Shows one tip of today with the option for the User to browse forward or backward to read more. Raises 'Magix.Core.GetPreviousToolTip' and 'Magix.Core.GetNextToolTip' to get its next and previous tips

Basetypes
Magix.Brix.Loader.IModule

Methods

public void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

44. Tree

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Tree

Description:
Level2: Shows a Tree module for the end user for him to navigate and select single nodes from. Change its properties by passing in 'TreeCssClass' or 'NoClose'. 'Items' should contain the tree items in a hierarchical fashion with e.g. 'Item/i-1/' containing 'Name' 'CssClass' and 'ToolTip'. 'Name' being minimum. Child items of 'Items/i-1' should be stored in the 'Items/i-1/Items' node. Will raise 'GetItemsEvent' upon needing to refresh for some reasons, and 'ItemSelectedEvent' with 'SelectedItemID' parameter as selected item ID upon user selecting an item

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Core.UpdateGrids")]
protected void Magix_Core_UpdateGrids(object sender, ActiveEventArgs e)
```

Level2: Overrides to make sure we also update this bugger upon changing of these types of records, but only if 'relevant'

```
[ActiveEvent(Name="Magix.Core.SetTreeSelectedID")]
protected void Magix_Core_SetTreeSelectedID(object sender, ActiveEventArgs e)
```

Level2: Sets the selected tree item to the 'ID' given

```
[ActiveEvent(Name="Magix.Core.ExpandTreeSelectedID")]
protected void Magix_Core_ExpandTreeSelectedID(object sender, ActiveEventArgs e)
```

Level2: Makes sure the 'currently selected tree item' becomes expanded, if any are selected

```
[ActiveEvent(Name="Magix.Core.UpdateTree")]
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, ActiveEventArgs e)
```

Level3: Overridden to handle some internal events

```
[ActiveEvent(Name="Magix.Core.GetSelectedTreeItem")]
protected void Magix_Core_GetSelectedTreeItem(object sender, ActiveEventArgs e)
```

Level2: Returns the selected tree item as 'ID' if any is selected

45. MessageBox

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.MessageBox

Description:
Level2: Implements logic for showing the end user a message box, asking for confirmation or something similar when needed. 'Text' will be the text shown to the user, which he should read and take a stand in regards to. Dropping the 'Cancel' Node sets Cancel to in-visible if you wish. 'OK/Event' will be raise with 'OK' node if OK button is clicked. 'Cancel/Event' will be raised with 'Cancel' node if Cancel button is clicked

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

46. Clickable

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Clickable

Description:
Level2: Encapsulates a Clickable Component, basically a button. Which you can load as a module, and trap clicks to. Set its 'Text', 'ButtonCssClass', 'Enabled' and 'ToolTip' params to customize it till your needs. Will raise 'Event' when clicked

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name="Magix.Core.EnabledClickable")]
protected void Magix_Core_EnabledClickable(object sender, ActiveEventArgs e)

Level2: Enables or disables [according to 'Enabled'] button

47. Header

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Header

Description:
Level2: Encapsulates a header [h1] control for you to create headers for your pages and apps. Load it and raise 'Magix.Core.SetFormCaption' to set the header

Methods

```
[ActiveEvent(Name="Magix.Core.UnlockFormCaption")]
protected void Magix_Core_UnlockFormCaption(object sender, ActiveEventArgs e)
```

Bad stuff. Refactor ...! To be removed ....

```
[ActiveEvent(Name="Magix.Core.SetFormCaption")]
protected void Magix_Core_SetFormCaption(object sender, ActiveEventArgs e)
```

Level2: Sets the caption of the header [h1] control

48. ImageModule

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.ImageModule

Description:
Level2: Control for displaying images in your app. Either clickable or static images. Pass in 'ImageURL', 'AlternateText', 'ChildCssClass' and 'Description' to modify it according to your needs. Description, if given, will add a label underneath the image. Use 'Seed' to have multiple Images on same page and to separate between different instances of them. If 'Events/Click' is given, it'll raise that event upon clicking the image

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Core.ChangeImage")]
protected void Magix_Core_ChangeImage(object sende, ActiveEventArgs e)
```

Level2: Updates the image to a new 'ImageURL'. But only if 'Seed' is given and correct according to 'Seed' given when loaded

49. ViewClassContents

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewClassContents

Description:
Level2: Will show all objects of type 'FullTypeName' in a Grid from which the user can filter, edit, change values, delete objects, create new objects and such from. Basically the 'show single table' logic of the Magix' 'Database Enterprise Manager'. If 'IsCreate', will allow for creation of objects of the 'FullTypeName' by the click of a button

Basetypes
Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

protected override void DataBindDone()

Level4: Called when databinding are done

```
[ActiveEvent(Name="Magix.Core.RefreshWindowContent")]
protected override void RefreshWindowContent(object sender, ActiveEventArgs e)
```

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

Properties

```
protected override System.Web.UI.Control TableParent
```

Level4: The control being our 'table element'

50. ViewListOfObjects

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewListOfObjects

Description:
Level2: Basically the same as ViewClassContents, though will only show objects 'belonging to a specific object [ParentID] through a specific property ['PropertyName'] and allow for appending, and not creation of new objects of 'FullTypeName'. Raises 'DBAdmin.Data.GetListFromObject' to get objects to display in Grid. Override the Append button's text property with 'AppendText'. Other properties are as normal from mostly every grid in Magix such as 'IsDelete', 'IsRemove' etc

Basetypes
Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

protected override void DataBindDone()

Level4: Called when databinding are done

Properties

protected override System.Web.UI.Control TableParent

Level4: The control being our 'table element'

51. ViewSingleObject

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewSingleObject

Description:
Level2: Contains the logic for editing and viewing one single ActiveType object, with all of its properties. Can become initiated in two different states, one of which is 'edit object reference from another object' which will allow for changing and removing the reference, the other is plain old 'edit the thing' mode. Supports 'ChildCssClass' and several other properties. Most of the common properties from the Database Enterprise Manager is included. Will raise 'DBAdmin.Form.ChangeObject' if user attempts to change the reference. Will raise 'DBAdmin.Data.RemoveObject' when object reference is removed. Changing the reference or removing it is only enabled if 'IsChange' and/or 'IsRemove' is given as true

Basetypes
Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name="DBAdmin.Form.CheckIfActiveTypeIsBeingSingleEdited")]
protected void DBAdmin_Form_CheckIfActiveTypeIsBeingSingleEdited(object sender, ActiveEventArgs e)

Level2: Will return 'Yes' == true if the given 'ID' matches the object being edited

```
[ActiveEvent(Name="DBAdmin.Form.ChangeCssClassOfModule")]
protected void DBAdmin_Form_ChangeCssClassOfModule(object sender, ActiveEventArgs e)
```

Level2: Will change the CSS class of the editing parts of the module if the 'FullTypeName' and the 'ID' matches. Useful for setting CSS class of specific 'Edit Object Module'

52. FindObject

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.FindObject

Description:
Level2: Basically identical to ViewClassContents, with one addition. This module will show a 'filter field' to the end user which the user can use to 'search for items' from within.

Basetypes
Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

protected override void DataBindDone()

Level4: Called when databinding are done

```
[ActiveEvent(Name="Magix.Core.RefreshWindowContent")]
protected override void RefreshWindowContent(object sender, ActiveEventArgs e)
```

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

Properties

```
protected override System.Web.UI.Control TableParent
```

Level4: The control being our 'table element'

53. BrowseClasses

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.BrowseClasses

Description:
Level2: Contains UI for letting the end user browse the ActiveType classes within his system as a Database Enterprise Management tool. Allows for seeings all classes in a Tree hierarchy and letting him select classes, which again will trigger editing of records in that class. Kind of like the Database Anterprise Management tool for Magix. Will raise 'DBAdmin.Data.GetClassHierarchy', which you can handle if you have 'meta types' you wish to display

Basetypes
Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

54. ConfigureColumns

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ConfigureColumns

Description:
Level2: Allows for editing of 'visible columns' per type level. Will show a form from which the end user can tag off and on the different columns he wish to see in his grid. Databinds towards 'WhiteListColumns' and raises 'DBAdmin.Data.ChangeVisibilityOfColumn' when visibility of column changes with 'Visible', 'FullTypeName' and 'ColumnName' as parameters

Basetypes
Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

55. ConfigureFilters

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ConfigureFilters

Description:
Level2: Will allow the end user to configure his filters, meaning what filter to filter each column by. There are many different types of filter in Magix. LIKE, Equal and so on. This form allows you to globally set the filters for specific types on specific columns of those types

Basetypes
Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

56. ShowClassDetails

[ActiveModule]

Magix.Brix.Components.ActiveModules.Documentation.ShowClassDetails

Description:
Level2: Doxygen helper class for displaying documentation about members of classes and such for our documentation system. Takes in 'FullName', 'Description' and so on. Will create a grid if displaying items according to the structure given

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

57. RichEdit

[ActiveModule]

Magix.Brix.Components.ActiveModules.Editor.RichEdit

Description:
Level2: Contains the UI for the RichEditor or WYSIWYG editor of Magix. That little guy resembling 'word'. Specify a 'SaveEvent' to trap when 'Text' is being edited.

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name="Magix.Brix.Core.GetRichEditorValue")]
protected void Magix_Brix_Core_GetRichEditorValue(object sender, ActiveEventArgs e)

Level2: Will flat out return the Value of the RichEditor. Does NOT support multiple editors on the same screen

58. Explorer

[ActiveModule]

Magix.Brix.Components.ActiveModules.FileExplorer.Explorer

Description:
Level2: Containe the UI for the Explorer component, which allows you to browse your File System on your server, through your browser. Basically a File System Explorer kind of control, which allows for renaming, deleting, and editing [to some extent] the files in your installation. Can be instantiated in Select mode by setting its 'IsSelect' input parameter. If 'CanCreateNewCssFile' is true, the end user is allowed to create a new default CSS file which he can later edit. 'RootAccessFolder' is the root of the system from where the current user is allowed to browse, while 'Folder' is his current folder. The control does some basic paging and such, and has support for will raise 'Magix.FileExplorer.GetFilesFromFolder' to get its items. The module will raise the value of the 'SelectEvent' paeameter when an item has been selected. The module supports browsing hierarchical folder structures

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

59. EditAsciiFile

[ActiveModule]

Magix.Brix.Components.ActiveModules.FileExplorer.EditAsciiFile

Description:
Level2: Kind of like Magix' version of 'Notepad'. Allows for editing of textfiles or text fragments, and allow for saving them

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

60. Slider

[ActiveModule]

Magix.Brix.Components.ActiveModules.Menu.Slider

Description:
Level2: Contains the UI for our SlidingMenu module, used in the administrator dashboard. Takes a recursive 'Items' structure containing 'Caption' and 'Event' which will be raised when clicked ['Event']

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

61. MetaView_Single

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaView.MetaView_Single

Description:
Level2: UI parts for showing a MetaView in 'SingleView Mode'. Basically shows a form, with items dependent upon the look of the view. This is a Publisher Plugin module. This form expects to be given a 'MetaViewName', which will serve as the foundation for raising the 'Magix.MetaView.GetViewData' event, whos default implementation will populate the node structure according to the views content in a Key/Value pair kind of relationship. This will serv as the foundation for the module to know which types of controls it needs to load up [TextBoxes, Buttons etc]Handles the 'Magix.MetaView.SerializeSingleViewForm' event, which is the foundation for creating new objects upon clicking Save buttons etc.This is the PublisherPlugin you'd use if you'd like to have the end user being able to create a new MetaObject

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name="Magix.MetaView.GetWebPartsContainer")]
protected void Magix_MetaView_GetWebPartsContainer(object sender, ActiveEventArgs e)

Level2: Will return the Container's ID back to caller [e.g. "content1"] if it's the correct WebPartTemplate Container according to the requested 'PageObjectTemplateID'

```
[ActiveEvent(Name="Magix.MetaView.SerializeSingleViewForm")]
protected void Magix_MetaView_SerializeSingleViewForm(object sender, ActiveEventArgs e)
```

Level2: Will serialize the form into a key/value pair back to the caller. Basically the foundation for this control's ability to create MetaObjects. Create an action, encapsulating this event, instantiate it and raise it [somehow] when user is done, by attaching it to e.g. a Save button, and have the form serialized into a brand new MetaObject of the given TypeName

```
[ActiveEvent(Name="Magix.Meta.Actions.EmptyForm")]
protected void Magix_Meta_Actions_EmptyForm(object sender, ActiveEventArgs e)
```

Level2: Will 'empty' the current form. Useful in combination with Save or Clear button

Properties

```
public   string   ViewName
```

Level2: The name of the MetaView to use as the foundation for this form

62. MetaView_Multiple

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaView.MetaView_Multiple

Description:
Level2: UI parts for showing a MetaView in 'MultiView Mode'. Basically shows a grid, with items dependent upon the look of the view. This is a Publisher Plugin module. Basically a completely 'empty' module whos only purpose is to raise the 'Magix.MetaType.ShowMetaViewMultipleInCurrentContainer' event, whos default implementation will simply in its entirety replace this Module, hence not really much to see here.This is the PublisherPlugin you'd use if you'd like to see a 'list of MetaObjects' on the page

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

Properties

public string ViewName

Level2: The name of the MetaView to use as the foundation for this form

63. LogInOutUser

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.LogInOutUser

Description:
Level2: PublisherPlugin containing most of the UI for allowing a user to login/out of your website. Can be set into both OpenID mode and 'only native mode' or both. Raises 'Magix.Publishing.GetStateForLoginControl' to determine the state of the module, meaning if it should show both OpenID logic and native logic or only one of them. Will raise 'Magix.Core.UserLoggedOut' if user logs out and 'Magix.Core.LogInUser' if the user tries to log in. The 'Magix.Core.LogInUser' default implementation again will raise 'Magix.Core.UserLoggedIn' if it succeeds. It'll pass in 'OpenID' if user has chosen to log in with OpenID and 'Username'/'Password' if user choses to login natively

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
[ActiveEvent(Name="Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Will return false if this webpart can just be 'reused' to the next page

```
protected void Magix_Publishing_GetTemplateColumnSelectTypeOfLoginControl(object sender,
ActiveEventArgs e)
```

Level3: Implementation of 'Get Select Type Of Login Control' for Magix. Will return a Select Lst back to caller

Properties

public string LoginMode

Level2: Which mode you wish to use for your login control. Legal values are 'OpenID', 'Native' and 'Both'. Signifying the obvious, both being the default which will allow you to log in either with username/password combination or an OpenID claim

64. Header

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.Header

Description:
Level2: A 'header' PublisherPlugin. Basically just an HTML h1 element

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Will return false if this webpart can just be updated by asking for new data

Properties

public string Caption

Level2: BAsically the content of the h1 element. Default value is 'The Caption of your Header'

65. TopMenu

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.TopMenu

Description:
Level2: PublisherPlugin containing a conventional 'File Menu' type of menu which normally is expected to find at the top of a page, which will 'drop down' child selection boxes for being able to select children. Useful for conventional applications, which should look like legacy code, or something. Takes the exact same input parameters as the SliderMenu PublisherPlugin

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Will return false if this webpart can just be 'reused' to the next page

66. SliderMenu

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.SliderMenu

Description:
Level2: PublisherPlugin containing the UI for the Sliding Menu Publisher Plugin. Meaning the default menu to the left in the front-web parts, which you can choose to inject into a WebPartTemplate in one of your Templates if you wish. Basically just loads the items once through raising the 'Magix.Publishing.GetSliderMenuItems' event, which should return the items as a 'Items' list, containing 'Caption', 'Event', 'Event' [Event name] and 'Event/WebPageURL' which normally will contain the page's URL

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Will return false if this webpart can just be 'reused' to the next page

67. Content

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.Content

Description:
Level2: A 'content' PublisherPlugin. Basically just a text fragment, that'll be edited through the Magix' RichText or WYSIWYG Editor

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

```
public override void InitialLoading(Node node)
```

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

```
[ActiveEvent(Name="Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)
```

Level2: Overridden to just 'fetch the text', since such things are easier and faster for the server and such than reloading the entire page. Will basically determine if the container and module name are the same, if they are, it'll return STOP which will stop the reloading of the new module. Then this module will raise 'Magix.Publishing.GetWebPartSettingValue' for the new 'WebPartID', which will return the new Text as 'Value'

Properties

public string Text

Level2: The actual content of the Content control ... ;)

68. ChildExcerpt

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.ChildExcerpt

Description:
Level2: Will show an excerpt of all its children, sorted with newest first, showing a maximum of PagesCount items. Kind of like a front page to a blog or a news website or something

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

Properties

public int PagesCount

Level2: How many of the latest pages will be shown in the excerpt

69. EditSpecificTemplate

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.EditSpecificTemplate

Description:
Level2: Allows for editing of WebPageTemplate objects. Contains most of the UI which you're probably daily using while adding and creating new templates and such

Basetypes
Magix.Brix.Loader.ActiveModule

70. EditSpecificPage

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.EditSpecificPage

Description:
Level2: Allows for editing of one single specific WebPage in the system. Contains most of the UI which you're probably daily using while adding and creating new pages and such

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name="Magix.Publishing.PageWasDeleted")]
protected void Magix_Publishing_PageWasDeleted(object sender, ActiveEventArgs e)

Level2: Needs to be handled here, in case it was 'this page'

[ActiveEvent(Name="Magix.Publishing.PageWasUpdated")]
protected void Magix_Publishing_PageWasUpdated(object sender, ActiveEventArgs e)

Page was somehow updated, and we need to reload the currently editing page

71. Sign

[ActiveModule]

Magix.Brix.Components.ActiveModules.Signature.Sign

Description:
Level2: A Signature module, basically a place where the end user can 'sign his name' to confirm a transaction of some sort. Will load up a big white thing, which can be 'drawn upon', together with two buttons, OK and Cancel which will raise the 'CancelEvent' and the 'OKEvent. 'OKEvent' will pass in 'Signature' being the coords for all the splines that comprises the Signature, which can be stored and later used as input to this Module, which will again load those splines in 'read-only mode'

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

72. Forum

[ActiveModule]

Magix.Brix.Components.ActiveModules.TalkBack.Forum

Description:
Level2: Basically a 'Forum module' which allows for posting and reading and replying to other people's opinions about 'whatever'. Not very good on its own, please use through TalkBack Controller to save head aches

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

73. Login

[ActiveModule]

Magix.Brix.Components.ActiveModules.Users.Login

Description:
Shows a login box with username/password/openid combo. Username/password has preference, but if only OpenID is given, the system will attempt at login you in using the OpenID claim. Note, this is not the Publisher Plugin login box. This is the 'main system one', which you get by going to ?dashboard=true

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

74. Settings

Magix.Brix.Components.ActiveTypes.Settings

Description:
Level2: Wrapper class for common/global settings within a Magix-Brix application. Use the overloaded this operator to access Settings through code

Methods

public void Reload()

Level3: Will reload all settings from DB ...

public void Clear()

Level4: Will delete all settings. Notice that this operation cannot be undone!

public void Remove(string key)

Level3: Removes the specific Setting Key form the collection

Properties

public Settings Instance

Level3: Retrieves the singleton instance of your settings. Reference the Settings ActiveTypes in your project, use the Instance property from below, and use the index operator of the class or Get/Set to access specific settings

public string this[string key]

Level3: Returns or sets the key value as a string

public IEnumerable< string > Keys

Level4: Returns all the keys which exists.

public int Count

Level3: Returns the number of settings in total in database

75. TipOfToday

Magix.Brix.Components.ActiveTypes.TipOfToday

Description:
Level3: Helper class for Tip [Today's Tips] logic

Methods

public void CreateTip(string tip)

Level3: Only way to create a tip form the outside

public string Previous(string seed)

Level3: Will return the 'previous' tip with the given seed

public string Next(string seed)

Level3: Will return the 'next' tip with the given seed

public string Current(string seed)

Level3: Will return the 'current' tip with the given seed, and not in any ways increament or decrement the 'currently viewed tips' like the sibling methods will

Properties

public TipOfToday Instance

Level3: Getter to gain access to today's tips

public int Count

Level3: Number of tips in database

76. LogItem

[ActiveType]

Magix.Brix.Components.ActiveTypes.Logging.LogItem

Description:
Class encapsulating log items in Magix. Magix has automated logging components which will log certain aspects about your system and its behavior. These log items are stored in your database through using this active type

Basetypes
Magix.Brix.Data.ActiveType-g

Properties

```
[ActiveField()]
public   DateTime        When
```

Automatically generated date of when log item was created

```
[ActiveField()]
public   string   LogItemType
```

A friendly 'type string' meant to easily allow filtering different categories of log items according to how the end user wants it

```
[ActiveField()]
public   string   Header
```

Friendly [and SHORT!] description of log item

```
[ActiveField()]
public   string   Message
```

More thorough description if feasable

```
[ActiveField()]
public   int      ObjectID
```

And integer value pointing to the ID of the 'relevant object', whatever that is

```
[ActiveField()]
public   int      ParentID
```

Similar to above. Just 'another container' for similar type of data

```
[ActiveField()]
public   string   StackTrace
```

StackTrace, if we're in an exception

```
[ActiveField()]
public   string   IPAddress
```

IP address of user triggering log item. Saved for security reasons

```
[ActiveField(IsOwner=false)]
public   UserBase      User
```

Logged in user triggering log item

```
[ActiveField()]
public   string   UserID
```

ID of cookie on disc for user, if any. Every request to Magix will automatically create a 'tracking cookie', upon from which the user will be automatically associated with, till he either uses a different client, or empties his cache somehow. This is the 'ID' of that tracking cookie. Which makes it possible to 'identify' users, even though they're not logged in

77. MetaObject

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaTypes.MetaObject

Description:
The storage for the 'Meta Application System' in Magix. Every time you create a new object, using a Meta View or something similar, a MetaObject is being created to hold this data. A Meta Object is basically an object 'without structure', where you can, by editing your views, maintain the structure you wish for your data. In such a way you can create your own Meta Applications, using Magix, without being imposed onto an existing data-structure in any ways

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Making sure every child has this as its parent to avoid 'cache bugs'. Plus other types of logic

public MetaObject Clone()

Will return a deep copy of the entire Meta Object, with all its Child Objects being cloned, and all its properties being cloned. WARNING; Might take insane amounts of time, if your object graph is huge

Properties

```
[ActiveField()]
public   string    TypeName
```

The 'type' of your MetaObject. This property is being extensively used through out the entire Magix to separate your MetaObjects into specific 'types' for you. By filtering upon these types, different types of objects can emerge, such as 'Customer' and 'Contacts' and such. Though to play well with others, you should probably prefix all your TypeNames somehow, with e.g. your Company Name or something, to avoid clashing with other modules and other people's/company's MetaObjects

```
[ActiveField()]
public   string    Reference
```

Normally an automatically generated reference of how the object came into existance. E.g. the name of the View that created it, etc

```
[ActiveField()]
public   DateTime         Created
```

An automatically maintained value signifying when the object was created

```
[ActiveField()]
public   LazyList< Property >     Values
```

All 'values', 'properties' or 'fields' of the object

```
[ActiveField(IsOwner=false)]
public   LazyList< MetaObject > Children
```

The child MetaObjects of this object. Every MetaObject can have as many children as it sees fit. You can this way have one Customer having several Contacts for instance. Later we will create possibilities within the MetaView system to react upon these, intelligently, somehow, and allow the user to 'nest forms' and such. But currently this is unresolved unfortunately. Please notice that these are 'loosely coupled objects', meaning they won't be deleted when parent is deleted, and they can belong to more than one object

```
[ActiveField(RelationName="Children", BelongsTo=true)]
public   MetaObject       ParentMetaObject
```

The parent of the this object, if any

78. Action

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaTypes.Action

Description:
Contains the serialized content of the Meta Action system in Magix. One Action is a wrapper around an Active Event, as described in the O2 Architecture Document. Basically, an Action can be thought of as a 'serialized version' of an ActiveEvent being raised, which you then can choose to raise whenever you wish, at the trigger of a button for instance. By creating your own actions, you can effectively implement your own type of logic and flow in Magix without having to code. Every Action must have a unique name, and is often either referenced by its Name or its ID property.

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Overridden to make sure name is unique, among other things

public Action Copy()

Will perform a deep copy of the Action, with all of its parameters, and return back to caller

Properties

```
[ActiveField()]
public   string    Name
```

Must be unique. Used to de-reference the action in other places. Normal coding convention should be to add at least your company name as the action's prefix or something, to avoid name clashings

```
[ActiveField()]
public   string    Description
```

Friendly description, shown to user, which should explain the action in 'human readable text' for the end user

```
[ActiveField()]
public   string    EventName
```

The Name of the ActiveEvent being raised by this action

```
[ActiveField()]
public   LazyList< ActionParams >          Params
```

The parameters to the ActiveEvent. Will become serialized as Nodes being passed into the Active Event

```
[ActiveField()]
public   bool      StripInput
```

If true, will CUT the incoming node, which might be useful to make sure your nodes doesn't grow into 'Kingdom Come'

```
[ActiveField()]
public   DateTime          Created
```

Automatically maintained by Magix to give you a date of creation for the record

79. MetaView

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaViews.MetaView

Description:
Contains the logic for our views in our 'Meta Application System'. A MetaView is something that defines what the user is supposed to see in regards to MetaObjects. Basically the UI 'representation' of your object graphs and properties. A MetaView can become the 'engine' for a WebPart through the MetaView_Single and MetaView_Multiple plugins for Magix

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Overridden to make sure Name is unique, among other things

public MetaView Copy()

Will perform a deep-copy on the entire MetaView with all its properties and return to caller

Properties

```
[ActiveField()]
public   string    Name
```

Name of MetaView

```
[ActiveField()]
public   string    TypeName
```

The 'Type Name' of the MetaObject associated with this view. If this is a SingleView for instance, it'll create exclusively types of the given TypeName. If it's a MultiView, it'll exclusively show items of this type. Take care when designing your MetaType hierarchy

```
[ActiveField()]
public   LazyList< MetaViewProperty >   Properties
```

All the properties for the MetaView

80. WebPageTemplate

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPageTemplate

Description:
Serves as a 'recipe' for a WebPage. Every WebPage is built from one of these. Contains the definition of which module to instantiate for instance, but not which parameters to send into it. Also contains the positioning of the WebParts and other common features, such as CSS classes and such. A Template is just that, a 'template' for your WebPages

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Delete()

Level3: Deletes the object from your data storage.

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

```
[ActiveField()]
public   string   Name
```

Friendly name of template. Serves no logical meaning. It is probably smart though to create a coding convention of some sort here

```
[ActiveField()]
public   LazyList< WebPartTemplate >   Containers
```

Our 'WebPart-recipes'. Contains the recipe for every webpart that should be added to our page. The number of WebPartTemplates, and their settings, will define how the WebPage's WebParts will look like. This property 'controls' how the WebParts property of the associated WebPage is being built

81. WebPart

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPart

Description:
Is one instance of a container on a WebPage. Baically one 'module' if you wish. A WebPage is created of several WebParts, each WebPart again is built from a corresponding WebPartTemplate associated with the WebPageTemplate the page itself is built upon. This class encapsulates the WebPart of this relationship, and hence serves as a wrapper for WebPart settings basically on a per page level

Basetypes
Magix.Brix.Data.ActiveType-g

Properties

```
[ActiveField(IsOwner=false)]
public   WebPartTemplate        Container
```

Which template [recipe] the WebPart is built upon

```
[ActiveField(BelongsTo=true)]
public   WebPage        WebPage
```

Which web page the WebPart belongs to

```
[ActiveField()]
public   LazyList< WebPartSetting >      Settings
```

Settings for this WebPart. Mostly contains 'values' for the ModuleSettingAttribute attributes of the plugin

82. WebPartTemplate

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPartTemplate

Description:
Serves as a 'recipe' for WebParts. Contains stuff such as positioning of webpart, and name of module to inject. Every WebPageTemplate has a list of these guys, which serves as recipe for how the WebPage's WebParts should be built, and which Plugins to load up

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Delete()

Level3: Deletes the object from your data storage.

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

```
[ActiveField()]
public   string    Name
```

Name of template. Serves no logical purpose. But please make meaningful names

```
[ActiveField()]
public   string    ViewportContainer
```

Which container in our ViewPort are we supposed to get stuffed into. Normally this value is being 'normalized' though, meaning the WebParts are stuffed out in chronological order, according to how they appear in the collection, up until we've either stuffed them all out, or ran out of Viewport Containers

```
[ActiveField()]
public   string    ModuleName
```

Contains the fully qualified name of the module to instantiate. Warning, if modifying this one 'by hand', bad things can happen! This shoulc contain the full name of the Module's C# Class, e.g.; 'MyCompany.MyComponent.MyModule'. These modules must be attributed as PublisherPlugins

```
[ActiveField()]
public   string    CssClass
```

Additional CSS class(es) to be injected into the webpart

```
[ActiveField(BelongsTo=true)]
public   WebPageTemplate       PageTemplate
```

Which page template our webpart template belongs to

```
[ActiveField()]
public   int       Width
```

Width in multiples of 30/70/110 [+40] units up til 24

```
[ActiveField()]
public   int       Height
```

Height in multiples of 18

```
[ActiveField()]
public   int       MarginLeft
```

Margin-Left in multiples of 18. Can go negative to -23

```
[ActiveField()]
public   int       MarginTop
```

Margin-Top in multiples of 18. Can go negative to -30

```
[ActiveField()]
public   int       MarginRight
```

'Padding' in multiples of 30/70/110 [+40] units up til 24. Can go negative to -23

```
[ActiveField()]
public   int       MarginBottom
```

Bottom-Top in multiples of 18. Can go negative to -30

```
[ActiveField()]
public   bool      Last
```

If true, will eliminate all margins [10px spacing] to the right of the webpart

```
[ActiveField()]
public   bool      Overflow
```

If true, will allow the element to overflow in the vertical direction. Meaning it will 'ignore' its height, and just 'flow as high as it is'

83. WebPageRoleAccess

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPageRoleAccess

Description:
Gives access to a page according to a specific role. This is the core class in Magix in regards to authorizing access to specific pages. Access in Magix is associated with a User, his Role and the WebPage the user tries to access. You cannot authorize on a more fine-grained level out than WebPages, out of the box. This class encapsulates the authorization logic in Magix. By default everyone have access to everything. If one Role is explicitly granted access, then all other roles will be denied access, unless they too are granted access. In addition access is 'inherited' from the 'Mother Page'. Meaning a child page will be default have the same access rights as its Mother page, unless the child page itself starts granting and denying access to specific pages

Basetypes
Magix.Brix.Data.ActiveType-g

Properties

```
[ActiveField(IsOwner=false)]
public   Role     Role
```

Role to grant users belonging to that role access to the Page in this object. All other roles, which are not having their own access objects towards the page, will now no longer have access to this page

```
[ActiveField(IsOwner=false)]
public   WebPage        Page
```

Page to grant users belonging to role access to. All other roles, which are not having their own access objects towards the page, will no longer have access to this page

84. OpenIDToken

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.OpenIDToken

Description:
Encapsulates an OpenID Token. Meaning an OpenID Claim. Magix has support for serving as both an OpenID Relying Party and an OpenID Provider. This class serializes the OpenID Claims for users when using Magix as an OpenID Relying Party, and associating these claims with specific users. This means you can associate an OpenID claim to your user and then use the OpenID Token to log into Magix later

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

```
public override void Save()
```

Overridden to make sure all Claims are unique within Magix

Properties

```
[ActiveField()]
public   string   Name
```

OpenID Claim or Token. E.g.; myGoogleUsername.blogspot.com

```
[ActiveField(BelongsTo=true)]
public   User     User
```

Which user the claim belongs to, if any

85. User

[ActiveType(TableName="docMagix.Brix.Components.ActiveTypes.Users.UserBase")]

Magix.Brix.Components.ActiveTypes.Publishing.User

Description:
Encapsulates one user in Magix' Publishing system. Inherited to add more, specific to the publishing system, to the class

Methods

public override void Save()

Overridden to make sure we've got a default AvatarURL before calling base

public new User SelectByID(int id)

Returns the object with the given ID from your data storage.

public new IEnumerable(User) SelectByIDs(params int[])

Returns a list of objects with the given ID from your data storage.

public new User SelectFirst(params Criteria[])

Returns the first object from your data storage which are true for the given criterias. Pass null if no criterias are needed.

```
public new IEnumerable( User ) Select(params Criteria[])
```

Returns all objects from your data storage that matches the given criterias. Pass null if you want all objects regardless of any values or criterias.

```
public new int CountWhere(params Criteria[])
```

Returns the number of objects in your data storage which is of type User where the given criterias are true.

Properties

```
[ActiveField()]
public   string   FullName
```

Full name of person

```
[ActiveField()]
public   string   AvatarURL
```

URL, can either be relative [media/images/myImage.jpg] or absolute [http://g.com/x.jpg]

```
[ActiveField()]
public   string   Phone
```

Phone number to person

```
[ActiveField()]
public   string   Address
```

Street Address to person

```
[ActiveField()]
public   string   City
```

City where person lives

```
[ActiveField()]
public   string   Zip
```

Zip, yup ...

```
[ActiveField()]
public   string   State
```

State [if US. Ignore if other countries]

```
[ActiveField()]
public   string   Twitter
```

The Twitter Handle to the person, e.g. 'WinergyInc'

```
[ActiveField()]
public   string   Facebook
```

The Facebook username to the person, e.g. 'polterguy'

```
[ActiveField()]
public   DateTime        BirthDate
```

Date of birth

```
[ActiveField()]
public   string   Country
```

Country where person lives

```
[ActiveField()]
public   string   Gender
```

Yup, normally 'Male' or 'Female'

```
[ActiveField()]
public   string   Language
```

Come estas? Comprende ...?

```
[ActiveField()]
public   string   Nickname
```

A small little personal 'nickname', e.g. 'polterguy'

```
[ActiveField()]
public   string   TimeZone
```

Which timezone is person within

```
[ActiveField()]
public   LazyList< OpenIDToken >        OpenIDTokens
```

List of OpenID Claims associated with person

```
public   new int Count
```

Returns the number of objects in your data storage which is of type T.

86. WebPage

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPage

Description:
Represents one web page in our system, or one unique URL if you wish. Contains a list of WebParts which again are areas of the screen within your web page. Every page must have at least one webpart, but can have many more. This class encapsulates the logic of 'Pages' in Magix, which is probably easy to understand within the context of publishing or CMS systems. Anyway, a page in Magix might also be a container for your application, plugins and WebParts. WebParts and Plugins again can be either your own creations through MetaViews or something similar, or actual C# code written plugins for your system. One page is often easy to understand if you can perceive it as 'one URL', while it is still much more powerful than any 'CMS Pages' or 'Publishing Pages' out there.Every page is built upon a 'recipe' which is the WebPageTemplate class. The WebPageTemplate class contains the logic for which plugin type it should ue etc, while the WebPage contains the settings for the type of plugins instantiated upon opening it

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

public override void Delete()

Level3: Deletes the object from your data storage.

Properties

```
[ActiveField()]
public   string    Name
```

Name of your Web Page. Serves nothing but as a friendly name and have no real meaning in the system. Doesn't need to be in any ways unique. Should be 'descriptive' such as if you've got a 'Header' type of WebPart, it might be smart to Name your webpart 'Header' too, or 'Sub-Section-Header' or something similar

```
[ActiveField()]
public   string    URL
```

Needs to be unique system wide, but will do its best at staying unique. Also serves as a Materialized Path for our Page Hierarchy. Please notice though that if you change the URL after the page is created, it'll update the URL's of ALL child pages, and their children again, and so on ifinitely inwards. This makes changing the URL after creating [many] children for it a potential very expensive operation

```
[ActiveField()]
public   DateTime         Created
```

Automatically kept track of. Keeps the 'created date' of the WebPage

```
[ActiveField(IsOwner=false)]
public   WebPageTemplate       Template
```

The Template the Page is built upon

```
[ActiveField()]
public   LazyList< WebPart >     WebParts
```

The parts, or containers of the Web Page

```
[ActiveField()]
public   LazyList< WebPage >   Children
```

Children Page objects. Every WebPage might have a list of children within it

```
[ActiveField(BelongsTo=true)]
public   WebPage         Parent
```

Parent web page. Is null if this is the top most page

87. Posting

[ActiveType]

Magix.Brix.Components.ActiveTypes.TalkBack.Posting

Description:
Level2: One Talkback posting. If Parent is null, this is a top-level posting, and the Children collection might have content. If not, it's a child itself of another top level posting, as in a 'reply'

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

```
public override void Save()
```

Level3: Overridden to set the User and When poperties

Properties

```
[ActiveField()]
public   string   Header
```

Level2: The descriptive header of the posting

```
[ActiveField()]
public   string   Content
```

Level2: The actual content of the posting

```
[ActiveField()]
public   DateTime        When
```

Level2: Automatically changed value of when posting was created

```
[ActiveField(IsOwner=false)]
public   UserBase        User
```

Level2: Which user created the posting

```
[ActiveField()]
public   LazyList< Posting >      Children
```

Level2: Only top-level postings have children. These children can be perceived as 'replies' to the 'OP' [Original Poster]

```
[ActiveField(BelongsTo=true)]
public   Posting Parent
```

Level2: If this is null, this is a top level posting, in which case it might have children

88. UserSettings

[ActiveType]

Magix.Brix.Components.ActiveTypes.Users.UserSettings

Description:
Level2: Settings stored on a 'per user level'. Useful for storing simple fact on a per user level

Basetypes
Magix.Brix.Data.ActiveType-g

Properties

```
[ActiveField()]
public  string  Name
```

Level3: The Name of the settings. Must be unique within the user. Normally you should prefix these kinds of 'glogal storages' with your company name or something, to avoid nameclashings

```
[ActiveField()]
public  string  Value
```

Level3: The Value, stored as string. Convert as you wish yourself

89. UserBase

[ActiveType(TableName="docMagix.Brix.Components.ActiveTypes.Users.UserBase")]

Magix.Brix.Components.ActiveTypes.Users.UserBase

Description:
Level2: A User is a single person using your website, being registered with a username and a password, and hence no longer 'anonymous'. A user is normally recognized through his 'Username', and he has a password which he can use to log into the system, normally. This class encapsulates that logic. PS! This class supports inheriting [though it's not really entirely stable quite yet ...!]

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public void RemoveSetting(string name)

Level3: Removes the given setting for the user

public override void Save()

Overridden to make sure username is unique etc.

public void SaveNoVerification()

Will save the user 'raw' running none of the verification logic to make sure the data is not violating some logic or something

```
public override void Delete()
```

Overridden to make sure we log this

```
public bool InRole(string roleName)
```

Returns true if user belongs to a role with the given name [case-in-sensitive search]

```
protected void SaveImpl(bool verify)
```

Implementation of our two other Save methods

Properties

```
[ActiveField()]
public   string   Username
```

Level2: Username. Normally used to login to the system with, in combination with the user's password

```
[ActiveField()]
public   string   Password
```

Level2: Password. Unique word, character combinations or a sentence which only the user knows about himself, and which is used to authenticate the user upon logging in

```
[ActiveField()]
public   string   Email
```

Level2: Email address of user

```
[ActiveField(IsOwner=false)]
public   LazyList< Role >          Roles
```

Level2: List of roles. Please notice that every user in Magix might belong to more than one role, though there's not UI currently for adding a user into more than one role, meaning defacto there's only one role per user today

```
[ActiveField()]
private  internal LazyList< UserSettings >          Settings
```

Level2: Settings for user instance

```
public   string   RolesString
```

Level2: Will return a string containing all roles user belongs to

```
public   UserBase        Current
```

Will return or change the currently logged in user object. Notice that Users are in Magix stored in cookies on the harddiscs of the browsers that clients visits your website with. These cookies are currently being stored as non-persistive, and HttpOnly, but will make sure that as long as user doesn't close his browser, he will become automatically logged in again upon visiting the site, if he had visited before without closing his browser

90. Role

[ActiveType(TableName="docWineTasting.CoreTypes.Role")]

Magix.Brix.Components.ActiveTypes.Users.Role

Description:
Level2: Contains the roles i the system. Every user belongs to a 'role' which gives him rights in regards to some aspect of functionality or something. All autorization, by default in Magix, will go through this Role class, and which specific roles the currently logged in user belongs to

Basetypes
Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Overridden to make sure Name is unique among other things

Properties

[ActiveField()]
public string Name

Level2: The name of the role, typically 'Administrator' or 'Guest' or something

91. Criteria

Magix.Brix.Data.Criteria

Description:
Level3: Abstract base class for all data storage retrieval criterias. Also contains several handy static constructors for easy creation of data storage retrieval criterias.

Methods

public Criteria Eq(string propertyName, object value)

Level3: Static constructor to create a criteria of type Equals.

public Criteria Ne(string propertyName, object value)

Level3: Static constructor to create a criteria of type NotEquals.

public Criteria Like(string propertyName, string value)

Level3: Static constructor to create a criteria of type LikeEquals. Notice that this is equal to the LIKE keyword from SQL, meaning you can use % and _ as control characters to look for 'wild card combinations'. Laymen terms; % == * and _ == ?

public Criteria Id(int id)

Level3: Static constructor to create a criteria of type CritID. Will create a Criteria that will demand the ID of the object is id

public Criteria NotId(int id)

Level3: Static constructor to create a criteria of type CritNoID. Will create a Criteria that will demand the ID of the object is NOT id

public Criteria NotLike(string propertyName, string value)

Level3: Static constructor to create a criteria of type LikeNotEquals. The opposite of Like

public Criteria Lt(string propertyName, object value)

Level3: Static constructor to create a criteria of type LessThen. Depends upon the column type, but normally a dash of reason can deduct this, regardless of the column type. String are compared alphabetically, all other from a 'least is less' standpoint. Will return only objects that are 'Less than' the given value

public Criteria Mt(string propertyName, object value)

Level3: Static constructor to create a criteria of type MoreThen. 'Opposite' of Lt. See Lt

public Criteria ParentId(int id)

Level3: Static constructor to create a criteria of type ParentIdEquals. Will enforce that the only objects being chosen are all belonging to a parent object, with the given id

public Criteria ExistsIn(int id)

Level3: Static constructor to create a criteria of type ExistsInEquals. Which will make sure there exists a relationship between the resulting object and the object with the given id in such a way that the other object are 'referencing' this one in one way or another. If you need to express the 'opposite', as in 'return objects who are referencing id object', then use the overloaded version of this method and pass in true as the reversed parameter. This will ensure that only objects which themselves are referencing the id object will be returned. For instance if a Customer has a LazyList of Contacts, you could find all Contacts referenced in a specific customer through using Contact.Select(Crieria.ExistsIn(customerId))If you want to go the other way, as in finding all Customers that are referencing a specific contact you'd have to go Customer.Select(Criteria.ExistsIn(contactIs, true))The above works, as long as the 'IsOwner' parts of a relationship equals false. If it's an 'IsOwner type of relationship', you'd rather using IsChild and IsParent methods

public Criteria ExistsIn(int id, bool reversed)

Level3: Static constructor to create a REVERSED criteria of type ExistsInEquals. See the documentation to the overload for an explanation

public Criteria HasChild(int id)

Level3: Static constructor to create a criteria of type HasChildId. Will only return objects that have the given id as their 'child objects' [meaning 'IsOwner' == true in its ActiveField declaration]

public Criteria Sort(string colName)

Level3: Static constructor to create a criteria of type Sort. Will sort on the Column name

```
public Criteria Sort(string colName, bool ascending)
```

Level3: Static constructor to create a criteria of type Sort. Will sort on the column name. ascending decides whether or not ascending or descending

```
public Criteria Range(int start, int end, string sortColumn, bool ascending)
```

Level3: Static constructor to create a criteria of type Range. Will first of all sort acording to the sortColumn, either ascending or descending depending upon the 'ascending' parameter. Then it will return only the [start, end} result set from that dataset. Executes actually __BLISTERING__ fast, even thought you probably wouldn't believe so ... ;)Basically the 'foundation' for the Grid System in many ways ...

Properties

```
public  string  PropertyName
```

Level3: Name of property associated with criteria.

```
public  object  Value
```

Level3: Value that criteria associates with the given property of your criteria.

92. Transaction

Magix.Brix.Data.Transaction

Description:
Level3: Implements transactional support for your updates and inserts. Use through the C# using keyword to get automatic rollbacks. Or implement finally yourself in your code. Remember to call 'Commit' before Transaction is lost, since otherwise. Caputt. Default is Rollback

Constructors

public Transaction(Adapter ad)
Level3: NOT meant for accessing directly

Methods

public void Commit()

Will Commit the entire batch of jobs, and release the transaction

protected void Rollback()

Level3: Will do a Rollback on your entire changes to the database. Will also reset the cache

Properties

public abstract IDbTransaction Trans

Level3: NOT meant for accessing directly

93. ActiveTypeAttribute

Magix.Brix.Data.ActiveTypeAttribute

Description:
Level3: Mark your well known types or entity types [or more correctly said; Active Types ;)] with this attribute to make them serializable. In addition you must inherit from ActiveType with the type of the type you're creating as the generic type argument. Notice that this attribute is for classes, you still need to mark every property that you wish to serialize with the ActiveFieldAttribute.

94. ActiveType-g

Magix.Brix.Data.ActiveType-g

Description:
Level3: Inherit your well known types or entity types - the types you want to serialize to your database from this class giving the generic argument type as the type you're creating. Notice that you also need to mark your types with the ActiveRecordAttribute attribute in addition to marking all your serializable properties with the ActiveFieldAttribute.

Basetypes
Magix.Brix.Data.TransactionalObject

Methods

public int CountWhere(params Criteria[])

Level3: Returns the number of objects in your data storage which is of type T where the given criterias are true

public T SelectByID(int id)

Level3: Returns the object with the given ID from your data storage. Will return null if either it's a 'type-clash' or object doesn't exist, which are for all practical concerns for you, and should be, irrelevant

public IEnumerable(T) SelectByIDs(params int[])

Level3: Returns a list of objects with the given ID from your data storage

public T SelectFirst(params Criteria[])

Level3: Returns the first object from your data storage which are true for the given criterias. Pass nothing () if no criterias are needed.

public IEnumerable(T) Select(params Criteria[])

Level3: Returns all objects from your data storage that matches the given criterias. Pass nothing () if you want all objects regardless of any values or criterias.

public bool operator!=(ActiveType(T, ActiveType(T)

Level3: Returns true if the two given objects does not have the same ID.

public bool operator==(ActiveType(T, ActiveType(T)

Level3: Returns true if the two given objects do have the same ID.

public void Delete()

Level3: Deletes the object from your data storage.

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

public override bool Equals(object obj)

Level3: Returns true if the given object have the same ID as the this object.

public override string ToString()

Level3: Will return the ID of the ActiveType. Little bit of 'debugging helping'

Properties

public override int ID

Level3: The data storage associated ID of the object. Often the primary key if you're using a database as your data storage.

public int Count

Level3: Returns the number of objects in your data storage which is of type T.

95. ActiveFieldAttribute

Magix.Brix.Data.ActiveFieldAttribute

Description:
Level3: Used to mark entity objects as serializable. If a property is marked with this attribute then it will be possible to serialise that property. Notice that you still need to mark you classes with the ActiveRecordAttribute. Also only properties, and not fields and such can be marked as serializable with this attribute.

96. Adapter

Magix.Brix.Data.Adapter

Description:
Level4: Abstract base class for all Database Adapters in Magix-Brix. If you wish to build your own data adapter then inherit from this class and implement the abstract methods, add up a reference to the dll and change the data-configuration line in your configuration file and it should work. Class provides some common functions needed for all database adapters like caching, instantiation and such in addition.

Methods

public void InvalidateCache()

Level4: Careful here. This is often a VERY expensive operation if you're doing it frequently. Though sometimes needed I guess. This one is mostly here for being able to go 'completely reset' in case of transactional rollbacks and such. If you use this one directly yourself, you'd better be sure you know what you're doing. Since alternatively you'd win the 'slowest system on the planet' award ... ;)

public abstract void ResetTransaction()

Level4: DO NOT TOUCH this one either. It's exclusively here [really] for internal usage in regards to transaction support for rollbacks and such

public object SelectByID(Type type, int id)

Level4: Retrieves an object with the given ID from your data storage.

```
public void Delete(int id)
```

Level4: Deletes the object with the given ID from your data storage.

```
public abstract int CountWhere(Type type, params Criteria[])
```

Level4: Should return the number of items of the given type from your data storage with the given criterias.

```
public abstract object SelectFirst(Type type, string propertyName, params Criteria[])
```

Level4: Should return the first object of type; "type" - with the given criterias.

```
public abstract IEnumerable( object ) Select(Type type, string propertyName, params Criteria[])
```

Level4: Should return all objects of the given type with the given criterias.

```
public abstract string GetConnectionString()
```

Level4: Should return some sort of string identification of the underlaying datasource

```
public abstract Transaction BeginTransaction()
```

Level4: Begins a new transaction object, which ensures the entire opertion within the scope of the transaction object will be either saved or rejected, and thrown an exception from ...

```
public abstract IEnumerable( object ) Select()
```

Level4: Should return all objects in your data storage.

```
public abstract void Save(object value)
```

Level4: Should save the given object into your data storage.

```
public abstract void Close()
```

Level4: Called when data storage should close. Often used to close file handles or database connections etc.

```
public abstract void Open(string connectionString)
```

Level4: Called when your data storage should be opened. Often used to open file handles or database connections etc.

```
protected abstract object SelectObjectByID(Type type, int id)
```

Level4: Should return the object from your data storage with the given ID being of the given Type.

```
protected abstract void DeleteObject(int id)
```

Level4: Should delete the object with the given ID from your data storage.

Properties

public AdapterInstance

Level4: Retrieves the configured database adapter. Notice that you would very rarely want to use this directly but instead access it indirectly through your ActiveType classes

public List< Type > ActiveTypes

Level4: A list of all your ActiveTypes in the system

public List< Type > ActiveModules

Level4: A list of all your ActiveModules in the system

public Dictionary< int, object > Cache

Level4: Dictionary of ID, ActiveType objects built up during the Request. Every ActiveType object being fetched during on HTTP request [one postback] is being held in a cache for the duration of the rest of that request on a 'per request basis'. This is because research have shown that even though completely different modules, unknown to each other really, are constantly running in paralel, they often tend to react upon the same actual list of objects. Meaning that by caching everything on one request like this we loose basically nothing, since we're in GC land anyway, and aren't really 'locking up memory' in anyways. While we also get to have a __BLISTERING__ fast DataAdapter technology, since after it's in the cache, the criterias will only execute to fetch the ID of the object you're requesting, if you're using 'complex queries', and if you're querying directly for the ID's, you'll get immediately here upon requesting your ActiveTypes. Hence; Hoahh ...! SWOSH ...! :P

97. TransactionalObject

Magix.Brix.Data.TransactionalObject

Description:
Baseclass for 'internal usage' behind ActiveTypes. Definitely candidate for refactoring. In general do NOT reference any members from here directly, except the ID of course

98. MSTransaction

Magix.Brix.Data.Adapters.MSSQL.MSTransaction

Description:
Level4: Implementation of Transaction class for MS SQL DataAdapter

Basetypes
Magix.Brix.Data.Transaction

Methods

public override void Commit()

Will Commit the entire batch of jobs, and release the transaction

protected override void Rollback()

Level3: Will do a Rollback on your entire changes to the database. Will also reset the cache

Properties

public override IDbTransaction Trans

Level3: NOT meant for accessing directly

99. MSSQL

Magix.Brix.Data.Adapters.MSSQL.MSSQL

Description:
Level4: Microsoft SQL Server Database Adapter, which probably works with 2005 and any later, for Magix-Brix. Contains all MS SQL specific logic needed to use Magix-Brix together with MS SQL. This is the default DataAdapter in use in Magix

Basetypes
Magix.Brix.Data.Internal.StdSQLDataAdapter

Magix.Brix.Data.IPersistViewState

Methods

public override void Open(string connectionString)

Level4: Opens a connection to the MS SQL database

public override Transaction BeginTransaction()

Level4: Will create a new Transaction and return to caller

public override int CountWhere(Type type, params Criteria[])

Level4: Should return the number of items of the given type from your data storage with the given criterias.

public override object SelectFirst(Type type, string propertyName, params Criteria[])

Level4: Should return the first object of type; "type" - with the given criterias.

public override IEnumerable(object) Select(Type type, string propertyName, params Criteria[])

Level4: Should return all objects of the given type with the given criterias.

public override IEnumerable(object) Select()

Level4: Should return all objects in your data storage.

public override void Save(object value)

Level4: Should save the given object into your data storage.

public override void Close()

Level4: Called when data storage should close. Often used to close file handles or database connections etc.

public void Save(string sessionId, string pageUrl, string content)

Level4: Called by Magix when it's time to save the ViewState. Normally expected to dump the stuff into a DB somewhere with the key of sessionId+pageUrl or something

public string Load(string sessionId, string pageUrl)

Level4: Called by Magix when it's time to reload the ViewState. Normally expected to dump the stuff out of a DB somewhere with the key of sessionId+pageUrl or something

public override void ResetTransaction()

Level4: DO NOT TOUCH this one either. It's exclusively here [really] for internal usage in regards to transaction support for rollbacks and such

public override string GetConnectionString()

Level4: Should return some sort of string identification of the underlaying datasource

protected override object SelectObjectByID(Type type, int id)

Level4: Should return the object from your data storage with the given ID being of the given Type.

protected override void DeleteObject(int id)

Level4: Should delete the object with the given ID from your data storage.

100. Helpers

Magix.Brix.Data.Internal.Helpers

Description:
Level3: Static helper class for data-storage Adapter developers. If yo're fiddling around here, you'd better know what you're doing ...!! ;)

101. StdSQLDataAdapter

Magix.Brix.Data.Internal.StdSQLDataAdapter

Description:
Level4: Common logic for all Database adapters that relies on standard SQL syntax. Wrappers for creating SQL text for derived data adapters that uses RDBS that relies on standard SQL syntax. Sorry, I have to prioritize what I comment. Needs refactoring though, that's one thing ..... but so does the entire Magix.Data namespace ... :(

Basetypes
Magix.Brix.Data.Adapter

102. ActiveModule

Magix.Brix.Loader.ActiveModule

Description:
Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your Modules, such as RaisingEvents etc. Inherit your ActiveModules from this class to simplify their usage

Basetypes
Magix.Brix.Loader.IModule

Methods

public void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

public Node RaiseSafeEvent(string eventName)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s). This method will trap any exceptions occuring, and show a message box back to the end user with its exception content, if any

protected Node RaiseEvent(string eventName)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s)

protected void RaiseEvent(string eventName, Node node)

Level3: Shorthand for raising events

protected bool RaiseSafeEvent(string eventName, Node node)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s). This method will trap any exceptions occuring, and show a message box back to the end user with its exception content, if any

protected void IncludeCssFile(string file)

Level3: Will include the given CSS file onto the page. Useful for injecting your own CSS files onto the page

protected string GetApplicationBaseUrl()

Level3: Will return the 'base' URL of your application. Meaning if your application is installed on x.com/f then x.com/f will always be returned from this method. Useful for using as foundation for finding specific files and so on

Properties

protected Node DataSource

Level3: The Node passed into InitialLoading will automatically be stored here ...

103. ActiveEvents

Magix.Brix.Loader.ActiveEvents

Description:
Level3: Class contains methods for raising events and other helpers, like for instance helpers to load controls and such. Though often you'll not use this directly, but rather use it through helper methods on your ActiveControllers and ActiveModules

Methods

public void RaiseLoadControl(string name, string position)

Level3: Loads a control with the given name (class name) into the given position (name of Magix.UX.Dynamic in the Viewport currently used). Use this method to load Modules. Notice that there exists an overload of this method which takes an object parameter that will be passed into the InitialLoading method when control is loaded.

public void RaiseLoadControl(string name, string position, Node parameters)

Level3: Loads a control with the given name (class name) into the given position (name of Magix.UX.Dynamic in the Viewport currently used). Use this method to load Modules. This overload of the method will pass the "initializingArgument" parameter into the InitialLoading method when control is loaded.

public void RaiseClearControls(string position)

Level3: Clear all controls out of the position (Magix-Dynamic) of your Viewport.

public void RaiseActiveEvent(object sender, string name)

Level3: Raises an event with null as the initialization parameter. This will dispatch control to all the ActiveEvent that are marked with the Name attribute matching the name parameter of this method call.

public void RaiseActiveEvent(object sender, string name, Node pars)

Level3: Raises an event. This will dispatch control to all the ActiveEvent that are marked with the Name attribute matching the name parameter of this method call.

public void CreateEventMapping(string from, string to)

Level3: Will override the given 'from' ActiveEvent name and make it so that every time anyone tries to raise the 'from' event, then the 'to' event will be raised instead. Useful for 'overriding functionality' in Magix. This can also be accomplished through doing the mapping in the system's web.config file

Properties

public ActiveEvents Instance

Level3: This is our Singleton to access our only ActiveEvents object. This is the property you'd use to gain access to the only existing ActiveEvents object in your application pool

104. ActiveModuleAttribute

Magix.Brix.Loader.ActiveModuleAttribute

Description:
Level3: Mark your Active Modules with this attribute. If you mark your Modules with this attribute you can load them using the PluginLoader.LoadControl method. This is the main attribute for being able to create ActiveModules

105. ActiveControllerAttribute

Magix.Brix.Loader.ActiveControllerAttribute

Description:
Level3: Mark your controllers with this Attribute. Notice that an Active Controller must have a default constructor taking zero parameters. This constructor should also ideally execute FAST since all controllers in your Magix-Brix project will be instantiated once every request.

106. ActiveController

Magix.Brix.Loader.ActiveController

Description:
Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your controllers, such as Loading Modules, raising events etc. Inherit your controllers from this class if you'd like to add more 'power' to them

Methods

protected delegate void executor()

Level3: Helper for executing 'dangerous code' such that if an exception happens, it'll 'swallow' the exception, and show a Message box showing the exception

protected Node LoadModule(string name)

Level3: Loads the given module and puts it into your default container

protected bool ExecuteSafely(executor functor, string msg, params object[])

Level3: Helper for execute code that you suspect might throw exceptions. Will trap exception and show a message box back to end user instead of allowing exception to penetrate through to Yellow Screen of Death. Will return true if operation didn't throw an exception and false if it did throw an exception

protected Node LoadModule(string name, string container)

Level3: Loads the given module and puts it into the given container. Will return the node created and passed into creation

protected void LoadModule(string name, string container, Node node)

Level3: Shorthand method for Loading a specific Module and putting it into the given container, with the given Node structure.

protected Node RaiseEvent(string eventName)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s)

protected void RaiseEvent(string eventName, Node node)

Level3: Shorthand for raising events.

protected void ShowMessage(string body)

Level3: Shows a Message to the user with the given body

protected void ShowMessage(string body, string header)

Level3: Shows a Message to the user with the given body and header

protected void ShowError(string body, string header)

Level3: Shows an Error Message to the user with the given body and header

protected void IncludeCssFile(string file)

Level3: Will include the given CSS file onto the page

protected string GetApplicationBaseUrl()

Level3: Will return the 'base' URL of your application

Properties

protected Page Page

Level3: Shorthand for getting access to our "Page" object.

107. ActiveEventAttribute

Magix.Brix.Loader.ActiveEventAttribute

Description:
Level3: Mark your methods with this attribute to make then handle Magix.Brix Active Events. The Name property is the second argument to the RaiseEvent, or the "name" of the event being raised. You can mark your methods with multiple instances of this attribute to catch multiple events in the same event handler. However, as a general rule of thumb it's often better to have one method handling one event

108. ActiveEventArgs

Magix.Brix.Loader.ActiveEventArgs

Description:
Level3: EventArgs class that will be passed into your Magix-Brix events - the methods you mark with the ActiveEvent Attribute. The Extra property will contain the "initializationObject" passed into the RaiseEvent.

Properties

public string Name

Level3: The name of the Active Event. Most Active Event Handlers will be mapped only to one Active Event, but ocassionally you'll have one Event Handler handling more than one Event. For cases like this the Name property might be useful to understand which event you're actually handling

public Node Params

Level3: This is the "initializationObject" passed into your RaiseEvent call. Use this parameter to pass around data between components

109. PluginLoader

Magix.Brix.Loader.PluginLoader

Description:
Level4: Helps load UserControls embedded in resources. Relies on that Magix.Brix.Loader.AssemblyResourceProvider is registered as a Virtual Path Provider in e.g. your Global.asax file. Use the Instance method to access the singleton object, then use the LoadControl to load UserControls embedded as resources. Kind of like the Magix' version of Page.LoadControl. Can be used directly by you, if you really know what you're doing though. In general, I'd say DON'T ...!!

Methods

public Control LoadActiveModule(string fullTypeName)

Level3: Dynamically load a Control with the given FullName (namespace + type name). This is the method which is internally used in Magix-Brix to load UserControls from embedded resources and also other controls. Since ActiveEvents might be mapped and overridden, you actually have no guarantee of that the event you wish to raise is the one who will become raised

Properties

public PluginLoader Instance

Level4: Singleton accessor. Allows access to the 'one and only' PluginLoader

```
public    List< Assembly >          PluginAssemblies
```

Level4: Will return all assemblies within your Application Pool, minus 'system assemblies'. Useful for 'meta stuff'

110. AssemblyResourceProvider

Magix.Brix.Loader.AssemblyResourceProvider

Description:
Level4: Helper class to make it possible to load controls (and more importantly) UserControls which are embedded as resources in DLLs. Not intended for direct usage, but will be in 'the background' and making sure you can load ActiveModules as resources from your DLLs

111. PublisherPluginAttribute

Magix.Brix.Publishing.Common.PublisherPluginAttribute

Description:
Level3: I'd be highly surprised if this is not your first entry to Magix in C#. This is the PublisherPlugin attribute, which you can use to create your own plugins for the Publishing system within. Implement this attribute on your ActiveModules and VOILA! They'll surface up as selections in your WebPageTemplate editing operations and be usable as plugins in your system. Probably the easiest way on the planet to create a plugin for any kind of system out there. Especially in combination with the logic behind the ModuleSettingAttribute

112. ModuleSettingAttribute

Magix.Brix.Publishing.Common.ModuleSettingAttribute

Description:
Level3: Wraps a setting property for a PublisherPlugin

113. PeriodCollection

Magix.Brix.Types.PeriodCollection

Description:
Level3: A collection class of Period types. Contains algebraic methods for OR, AND, XOR, NOT. Makes algebraic operations on collection of Period objects very easy and intuitive. Can with for instance one line of code OR two collections together to find the logically OR'ed result of these two different collections. Very useful for manipulating dates and such

Constructors

public PeriodCollection()
Level3: Default CTOR

public PeriodCollection(IEnumerable(Period)
Level3: Initializes the list with the given collection

Methods

public void Normalize()

Level3: Sorts and runs through all period objects and checks for overlapping, eliminating overlapping periods merging them into one and such. Kind of like the same mathematical operation as normalizing a vector. Crucial for most of the algebraic operations. This operation will most often make the containing number of items less then it used to be before the operation. Meaning two Periods that overlap will become one 'combined' period after this method is done

public PeriodCollection NOT()

Level3: Returning the NOT operation of the this list. Basically all the places where there are NO Periods within the collection of Periods. Kind of the "negative" of the collection. Notice that since we do NOT threat a PeriodCollection as an "open end collection" but rather as a min value of DateTime.MinValue and a max value of DateTime.MaxValue we do not get the "elephant footstep" as we would otherwise be forced to have. Meaning that the operation is 'reversible' by calling it twice.

public TimeSpan Sum()

Level3: Returns the total amount of time in the this collection

public void Trim(Period period)

Level3: Trimming away everything that's not within given Period

public void AddRange(IEnumerable(Period)

Level3: Adds the given range of periods into the collection

public void Sort(Comparison(Period)

Level3: Takes a predicate and sort the list accordingly

public void Sort()

Level3: Sorts according to a predicate that sorts first prioritized after start of periods and then according to end of periods if start of periods are the same. Used in the Normalize method

public Period Find(Predicate(Period)

Level3: Takes a predicate and returns the first Period that matches the predicate

public List(Period) FindAll(Predicate(Period)

Level3: Returns a list of Periods that matches the predicate

public void RemoveAll(Predicate(Period)

Level3: Removes all the periods that matches the predicate

public int IndexOf(Period item)

Level3: Returns the index of the given item, or -1 if no found

public void Insert(int index, Period item)

Level3: Inserts the given period at the given index

public void RemoveAt(int index)

Level3: Removes the period at the given index

public void Add(Period item)

Level3: Appends to the back of the list the given period.

public void Clear()

Level3: Completely clears all periods from the list

public bool Contains(Period item)

Level3: Returns true if the given period is found in the list

public void CopyTo(Period[] array, int arrayIndex)

Level4: Copy the list of periods into the given array starting at arrayIndex in the collection

public bool Remove(Period item)

Level3: Removes the given period from the collection

public IEnumerator(Period) GetEnumerator()

Level3: Enumerating support

public PeriodCollection OR(PeriodCollection left, PeriodCollection right)

Level3: Returns the logically ORed lists back to caller. Does not in any ways change the given lists.

public PeriodCollection AND(PeriodCollection left, PeriodCollection right)

Level3: Logically ANDs two collections together returning the ANDed result. Basically a new collection of Period objects which consists of the date ranges that overlaps within both lists. Useful for finding out for instance when two period collections overlap

public PeriodCollection XOR(PeriodCollection left, PeriodCollection right)

Level3: Returning the logical XOR of two given collections. The XOR result is the place where ONE and ONE ONLY of the given collections have values. This will normally increase the number of periods in your collection. Doesn't change the incoming periods [left, right]

Properties

public DateTime Starts

Level3: Returns the lowest start date of the collection

public DateTime Ends

Level3: Returns the highest end date of the collection

public Period this[int index]

Level3: Returns the period at the given index. Will throw if out of bounds. Setter will replace the existing period at the given index.

public int Count

Level3: Returns the number of items in the collection

114. LazyList-g

Magix.Brix.Types.LazyList-g

Description:
Level3: Helper class for Lazy Loading of Child ActiveTypes objects. Basically just a list generic type, that'll not load objects before needed. Useful for using as properties for list of child objects in your ActiveTypes

Methods

public bool Exists(Predicate(T)

Level3: Traverses the list and returns true if the item you're looking for exists, as in the predicate returns true

public T Find(Predicate(T)

Level3: Traverses the list and returns the first item matching the predicate

public IEnumerable(T) FindAll(Predicate(T)

Level3: Traverses the list and returns all items matching the predicate

public void AddRange(IEnumerable(T)

Level3: Adds a range of new items to the list collection

```
public void RemoveAll(Predicate( T)
```

Level3: Removes all items matching the given predicate

```
public int IndexOf(T item)
```

Level3: Returns the index of the given item, if it exists in the list

```
public void Insert(int index, T item)
```

Level3: Inserts a new item at the given position

```
public void RemoveAt(int index)
```

Level3: Removes the item at the given index

```
public void Add(T item)
```

Level3: Appends a new item to the list

```
public void Clear()
```

Level3: Clears the list of all its items

public bool Contains(T item)

Level3: Returns true if the specific item exists in the list

public void CopyTo(T[] array, int arrayIndex)

Level4: Copies the list to the given array, starting at the given offset

public bool Remove(T item)

Level3: Removes the specific item from the list, returns true if an item was removed. Returns false if the item doesn't exist in the list

Properties

public bool ListRetrieved

Level3: Returns true if list is already populated. If the LazyList is not retrieved, then the child objects in that property won't be saved when saving the parent object. Which can be very useful for optimizing your application. Be careful with 'de-referencing' LazyList properties because of this, unless you really have to

public T this[int index]

Level3: Gets or sets the item at the specific index

```
public   int      Count
```

Level3: Returns the number of items in our list

```
public   bool     IsReadOnly
```

Will return false, always!

115. Node

Magix.Brix.Types.Node

Description:
Level3: Helper class to pass around data in a "JSON kind of way" without having to convert to JSON strings. Create a new instance, and just start appending items to it like this; Node n = new Node(); n["Customer"]["Name"] = "John Doe"; n["Customer"]["Adr"] = "NY"; This is at the core of Magix, being the 'protocol' we're using to pass data around within the system. If you don't understand this class, you're in trouble! Make sure you understand, at least roughly, what this class does if you'd like to code C# for Magix

Constructors

public Node()
Default CTOR, creates a new node with no name and no value and no children

public Node(string name)
Creates a new node with the given name

public Node(string name, object value)
Creates a new node with the given name and the given value

Methods

public Node Find(Predicate(Node)

Level3: Returns the first node that matches the given Predicate. Will search recursively. Be creful here, if you're dereferncing nodes that don't exist inside your function, you might very well never return from this method ... ;)

public bool Exists(Predicate(Node)

Level3: Returns true if node exists as a direct child only, and not search recursive

public bool ExistsDeep(Predicate(Node)

Level3: Returns true if node exists anywhere as a child or child of child

public Node UnTie()

Level3: Will "disconnect" the node from its parent node. Ueful for removing nodes and trees out of Node structures

public int IndexOf(Node item)

Level3: Returns the index of the given item, if it exists within the children collection. Otherwise it returns -1

public void Insert(int index, Node item)

Level3: Inserts a new item into the children collection

public void RemoveAt(int index)

Level3: Removes node at given index

public void Add(Node item)

Level3: Adds a new node to the collection

public void AddRange(IEnumerable(Node)

Level3: Adds a range of nodes to collection

public void Clear()

Level3: Entirely empties the collection

public bool Contains(Node item)

Level3: Returns true if node exists within child collection [flat]

public bool Contains(string itemName)

Level3: Returns true if node exists within child collection [flat]

public bool Remove(Node item)

Level3: Removes the given node from the child collection

public IEnumerator(Node) GetEnumerator()

Level3: Supports enumerating items

public override string ToString()

Level4: Will return name/value and number of children as a string

public string ToJSONString()

Level3: Will translate the Node structure to a JSON string. Useful for passing stuff around to other systems, and integrating with client-side etc. Be warned! No TYPE information is being passed, so you cannot build the same Node structure by reversing the method and call FromJSON after creating JSON out of your node

public void Sort(Comparison(Node)

Level3: Will sort the nodes according to your given comparison delegate

public Node FromJSONString(string json)

Level3: Will de-serialize the given JSON string into a Node structure. PS! Even though Nodes can be serialized to JSON, the type information is lost, meaning you can not go back again 100% correctly, since you're 'loosing your types' when converting from Node to JSON. This is on our road map for fixing, but currently not finished

Properties

public Node Parent

Level3: Returns the Parent node of the current node in the hierarchy. Normally you'd never need to know the 'Parent' of a node, due to the intrinsic logic of Magix, so be careful. If you're using this method, you're probably doing something wrong on an architectural level. Be warned ...

public string Name

Level3: Returns the name of the node

public object Value

Level3: Returns the value of the object

public Node this[string name]

Level3: Returns the node with the given Name. If that node doesn't exist a new node will be created with the given name and appended into the Children collection and then be returned. Please notice that this method CREATES NODES during DE-REFERENCING. And it is its INTENT TOO! ;)

public Node this[int index]

Level3: Returns the n'th node

public int Count

Level3: Returns the number of items in the children collection

116. SingleContainer

[ActiveModule]

Magix.Brix.Viewports.SingleContainer

Description:
Level2: Contains the logic for the main Viewport in Magix. A viewport can be seen as your 'design' and contains all the different logic for being able to load and unload modules and such

Basetypes
Magix.Brix.Loader.ActiveModule

Methods

private void IncludeCssFile(string cssFile)

Level3: Will include the given CSS file onto the page. Useful for injecting your own CSS files onto the page

[ActiveEvent(Name="")]
protected void NULLEventHandler(object sender, ActiveEventArgs e)

Level2: Handled to make sure we log our events int the Debug window if enabled

[ActiveEvent(Name="Magix.Core.SetTitleOfPage")]
protected void Magix_Core_SetTitleOfPage(object sender, ActiveEventArgs e)

Level2: Will set the Title element of the page to the given 'Caption'

```
[ActiveEvent(Name="Magix.Core.SetViewPortContainerSettings")]
protected void Magix_Core_SetViewPortContainerSettings(object sender, ActiveEventArgs e)
```

Level2: Will change th Viewport's settings such as CSS class, margins, size etc. Legal parameters are 'Width', 'Top', 'MarginBottom', 'PullTop', 'Height', 'PushLeft', 'PushRight', 'Padding', 'Last' and 'CssClass'

```
[ActiveEvent(Name="Magix.Core.GetViewPortSettings")]
protected void Magix_Core_GetViewPortSettings(object sender, ActiveEventArgs e)
```

Level2: Will return the settings for the Viewport back to caller. 'Width', 'Top', 'MarginBotto', 'Last' and so on

```
[ActiveEvent(Name="Magix.Core.AddLinkInHeader")]
protected void Magix_Core_AddLinkInHeader(object sender, ActiveEventArgs e)
```

Level2: Will add a 'link header element' to your rendered HTML

```
[ActiveEvent(Name="Magix.Core.AddCustomCssFile")]
protected void Magix_Core_AddCustomCssFile(object sender, ActiveEventArgs e)
```

Level2: Injects a CSS file onto the page for inclusion on the client side for you

```
[ActiveEvent(Name="Magix.Core.ShowMessage")]
protected void Magix_Core_ShowMessage(object sender, ActiveEventArgs e)
```

Level2: Will show a 'Message Box' with your 'Message', 'Header' for 'Milliseconds' time period. If 'IsError' is true, it'll be red and contain some 'error logic' within it. If 'Delayed' is true, the message will not be shown directly, but in fact 'postponed' to the next request. Which canbe useful for e.g. Async event handlers, needing to tell the user something, or when you're redirecting the user, but need to explain him why and such

```
protected void ClearControls(object sender, ActiveEventArgs e)
```

Level2: Will clear the incoming 'Position' container for controls, and unload and clean up everything in regards to any modules within that container

```
[ActiveEvent(Name="Magix.Core.GetNumberOfChildrenOfContainer")]
protected void Magix_Core_GetNumberOfChildrenOfContainer(object sender, ActiveEventArgs e)
```

Level2: Will return the number of Active Modules [or controls] a specific Viewport Container contains. Useful for determining of a specific container is available or not

```
[ActiveEvent(Name="Magix.Core.LoadActiveModule")]
protected void Magix_Core_LoadActiveModule(object sender, ActiveEventArgs e)
```

Level2: Handled to make it possible to load Active Modules into this Viewport's containers

```
[ActiveEvent(Name="Magix.Core.SetAjaxWaitImage")]
protected void Magix_Core_SetAjaxWaitImage(object sender, ActiveEventArgs e)
```

Allows you to change the 'Ajax Wait Image' for your application. Every time an Ajax request [something is clicked e.g.] is sent to the server, a Please Wait "Window" will display an animated image while you're waiting. If you wish to change this animated Image, you can raise the 'Magix.Core.SetAjaxWaitImage' event, which will update the image according to the new 'Image' value passed in

117. AspectModal

Magix.UX.Aspects.AspectModal

Description:
Aspect useful for making 'modal' controls. A Modal control will obscur the contents behind it by creating a div that is semi-transparent (most often) and fill the entire viewport (browser area). This effect makes it impossible to click controls behind it on the page.

Basetypes
Magix.UX.Aspects.AspectBase

Methods

protected override string GetClientSideScript()

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

Properties

public decimal Opacity

The transparency of the control. If this value is 1 the 'modality-div' created will be 100% opaque. If 0.5 it will be 50% transparent, if 0 it will be 100% transparent, but still actually do its job to some extent.

public Color Color

The color of the 'modality-div'. The default value is 'Black'. If you also set its BottomColor property, then a gradient will be created between the Color (top parts) and the BottomColor (bottom) and it will gradient between those starting from the top.

public Color BottomColor

The color of the 'modality-div'. The default value is 'Black'. If you also set its BottomColor property, then a gradient will be created between the Color (top parts) and the BottomColor (bottom) and it will gradient between those starting from the top.

118. AspectAjaxWait

Magix.UX.Aspects.AspectAjaxWait

Description:
Ajax Wait Aspect. Use this one whenever you've got an Ajax Control that you know will spend a lot of time on the server-side executing its logic. You can either have the client-side logic create its own DOM element to serve as a 'blackout' when it's supposed to display the Ajax Wait DOM element, or you can attach an existing in-visible DOM element to it by using the Element property.

Basetypes
Magix.UX.Aspects.AspectBase

Methods

protected override string GetClientSideScript()

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

Properties

public int Delay

The number of milliseconds that will pass after an Ajax Request has been initiated before the Ajax Wait obscurer will be shown. The default value is 500.

public decimal Opacity

The opacity or transparency of the obscurer DOM element. The default value is 0.5 which means 50% transparent.

public string Element

The Element you've attached to this particular Ajax Wait. If none is given, Magix UX will on the client-side automatically create a black DOM element for you which will serve as the obscurer.

119. AspectBase

Magix.UX.Aspects.AspectBase

Description:
Base class for all Aspects in Magix UX. Implements common functionality.

Basetypes
Magix.UX.Widgets.Core.BaseControl

Methods

protected override void RenderJson(HtmlTextWriter writer)

Only override this one if you truly know what you're doing, and you really need some modified logic. See how the Aspects work for a working example of how this method can be overridden.

public void EnsureViewStateLoads()

Helper methods for Aspect developers. Loads state from the client-side into the Control/Aspect. Used in e.g. AspectDraggable for changing the top/left property of the control when moved and its got a server-side binding (event handler)

120. AspectDraggable

Magix.UX.Aspects.AspectDraggable

Description:
Aspect for creating controls that can be moved around on the screen using your mouse. It is also possible to handle the Dragged event and track when the dropping of the control occurs.

Basetypes
Magix.UX.Aspects.AspectBase

Methods

public override void EnsureViewStateLoads()

Helper methods for Aspect developers. Loads state from the client-side into the Control/Aspect. Used in e.g. AspectDraggable for changing the top/left property of the control when moved and its got a server-side binding (event handler)

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler Dragged
Handle this event to track when the control is dragged and dropped by the user. If this one is ommitted, there will not be raised an Ajax Request at all when the control is moved, and hence the ViewState value of the top/left corner of the control will never be updated.

Properties

public Rectangle Bounds

The bounding rectangle that the control is possible to move within. Rectangular constraints. The default is 'no constraints'.

public Point Snap

The grid that the control will be moved within. If this one is set to 50,50 then the control will snap into a 'grid' of 50 pixels large areas. Useful for making sure a control is being moved only within some rectangular constraints.

public string Handle

The handle from which the control itself will be able to drag from. If ommitted you can drag the control from any place inside of the control. If a handle is defined, then the control can only be dragged from this specific handle. The Window control is a good example of utilizing this logic. You can drag the entire Window, which means the Window has an AspectDraggable associated with it. But you can only drag the Window by its Caption, which is the handle for the AspectDraggable within the Window.

public bool Enabled

If this one is false, then the control cannot be dragged. Useful for having an AspectDraggable which might be enabled or not on the fly. The Window is a good example of this. It always have the AspectDraggable associated with it, but if the Window is being set to 'non-movable', then the AspectDraggable will be disabled.

121. SlidingMenu

Magix.UX.Widgets.SlidingMenu

Description:
This widget is a 'menu looka-like kinda control'. Its purpose is to be an alternative way of displaying hierarchical choices for your users. But instead of 'popping up' like menu items does in a conventional menu, they will 'slide in' from the side, making a much more efficient way of handling your space on your web pages. For most navigational purposes, this widget is a far better bet than the conventional Menu widget.

Basetypes
Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler MenuItemClicked
Raised when any SlidingMenuItem is clicked within the SlidingMenu hierarchy. Notice that if you actually choose to create an event handler for this event, then anything you do will trigger a server-request, and hence your application will not be as responsive. You might, in most cases, choose to only implement event handlers for the LeafMenuItemClicked instead of handling this event handler.

public EventHandler LeafMenuItemClicked

Raised when a 'leaf menu item' is clicked. A Leaf MenuItem is a menu item without any children. Since most menu items which have children are used as groupings, and not really all that interesting to get notifications about when clicked, this is probably the event you wish to handle, unless all your menu items are URL menu items, in which case you don't even need to handle this event.

Properties

public string ActiveMenuItem

The active SlidingMenuItem of your SlidingMenu. Use this one to see which menu item was clicked in for instance your event handlers for menu item clicked.

122. SelectList

Magix.UX.Widgets.SelectList

Description:
A DropDown list type of control, although it can also be set into a non-drop down mode. Basically multiple choices type of widget. To some extent, it overlaps the logical functionality of the RadioButton widget, although the SelectList is more useful for cases where you have a massive number of choices, like for instance choose one out of 300 different languages, while the RadioButton is more useful for cases where you have fewer choices, such as choose 'coffee, tea or water'. Add up your choices by adding up ListItems inside of your SelectList.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Events

public EventHandler SelectedIndexChanged
Raised whenever the selected value of your select list is changed. You can also set the selected value in code through using for instance the SelectedIndex property. Whenever a user changes the actively selected item of your select list, this event will be raised.

Properties

public int Size

If this property is anything higher than 1, which is the default value, the select list will no longer be a 'drop down' type of multiple choices widget, but will show as a 'panel' showing all the choices and making it possible to scroll, if needed, to see more items further down. This property is the number of items that the select list will show at one time. Anything higher than 1, will change its appearence completely and make it into a non-drop-down type of widget.

public ListItemCollection Items

A collection of all the items you have inside of your select list. This will be automatically parse through your .ASPX syntax if you declare items inside of the .ASPX file, or you can also programmatically add items in your codebehind file.

public ListItemSelectedItem

Will return the currently selected item or set the currently selected item. There are multiple duplicates of this property, like for instance the SelectedIndex property. The default SelectedItem will always be the first (zero'th) element, regardless of which property you're using to retrieve it.

public int SelectedIndex

Will return the index of the currently selected item or set the currently selected item based on its index. There are multiple duplicates of this property, like for instance the SelectedItem property. The default SelectedItem will always be the first (zero'th) element, regardless of which property you're using to retrieve it. Meaning that the default value of this property will always be '0'.

123. SlidingMenuLevel

Magix.UX.Widgets.SlidingMenuLevel

Description:
Child control of a SlidingMenuItem. Will contain the SlidingMenuItems of a SlidingMenuItem. If you have items which have children, then you must create an item of type SlidingMenuItem and add those children inside of that object. This is true both for .ASPX markup and SlidingMenu hierarchies created in code.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

124. SlidingMenuItem

Magix.UX.Widgets.SlidingMenuItem

Description:
A single sliding menu item. One instance. A SlidingMenu are basically composed out of a whole hierarchy of these types of widgets.

Basetypes
Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The (visible) text of your menu item.

public string URL

If you give this property a value, the item will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

public SlidingMenuLevel SlidingMenuLevel

Will return the SlidingMenuLevel of the MenuItem, if any exists. Every item that have child items, should create a SlidingMenuLevel and put its sub items within the SlidingMenuLevel portion of their markup, or instantiate them within the SlidingMenuLevel child control of those items, if the items are being added programmatically.

private SlidingMenu SlidingMenu

Will return the parent SlidingMenu of this item. This property will traverse upwards in the ancestor hierarchy and find the first SlidingMenu instance, being an ancestor of this particular item and return that object.

125. MultiPanelView

Magix.UX.Widgets.MultiPanelView

Description:
Instances of items within a MultiPanel. These are the items a MultiPanel is made of. Most scenarios you would use this control within would have only one of the MultiPanelViews visible at the same time.

Basetypes
Magix.UX.Widgets.Panel

126. MultiPanel

Magix.UX.Widgets.MultiPanel

Description:
A MultiPanel is a collection of MultiPanelViews, often used to create TabControls or similar constructs. Often combined with TabStrip and TabButton. This control can be configured extensively to mimick a lot of different scenarios. Hence we don't call it TabControl, although that happens to be one of the sub-scenarios you can create with it.

Basetypes
Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Methods

public void SetActiveView(int index)

Sets the Active MultiPanelView of this instance according to the index passed in.

public void SetActiveView(MultiPanelView selectedView)

Sets the Active MultiPanelView of this instance according to the MultiPanelView passed in.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public int AnimationDuration

The Speed of the Animation when switching active view.

public int ActiveMultiPanelViewIndex

Active/visible view. Will only work if MultiButtonClicked event handler is defined for MultiButton, or you run your own "change logic"...

public AnimationType AnimationMode

How to animate AccordionViews when opened, see the AnimationType for details.

127. RadioButton

Magix.UX.Widgets.RadioButton

Description:
A two-state widget type that can be grouped together with other widgets to mimick multiple selections. Like for instance; "Would you like to have coffee, tea or water" is a perfect example of where you would want to use RadioButtons. If you want to group specific radio buttons together, you must give them the same GroupName property. If you do, then only one of these RadioButtons can at any time be 'selected'.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler CheckedChanged
Event raised when the checked state of the widget changes. Use the Checked property to determine if the CheckBox is 'on' or 'off'.

Properties

public string GroupName

If multiple RadioButtons are given the same GroupName, then these RadioButtons will be grouped together, and only one of them can at any time be selected. By playing with this property, you can create multiple groups of radio buttons that will be grouped together accordingly to their GroupName property.

public bool Checked

Use this property to determine if the widget is checked or not. If this property is true, then the widget is checked. The default value is 'false'. You can also set this value in your code to change the state of the checked value.

128. Panel

Magix.UX.Widgets.Panel

Description:
A container widget for displaying other widgets inside of it. Will render as a div by default, but the specific tag this widget will render, can easily be overridden by changing the Tag property. You can choose to render your panels as paragraphs (p...) for instance. If you only need to display text on your page, and you need to use WebControls for this, you should use the Label control and not the Panel control.

Basetypes
Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

Properties

public string Tag

The HTML tag element type used to render your widget. You can set this property to anything you wish, including 'address', 'p' or any other types of HTML tags you wish to use to render your widget. If you need an inline-element, such as a span or something, or you need to render a widget without child widgets, you should rather use the Label widget.

public string DefaultWidget

The default widget which will be mimicked a 'click' when this widget has focus on the user clicks carriage return (Enter/Return key) on his keyboard. Useful for information typing where you don't want to force the user of clicking a specific 'save' button. If you use another widget as the default widget, you must remember to use the ClientID property of that widget as the DefaultWidget value of this property.

129. TreeItem

Magix.UX.Widgets.TreeItem

Description:
One single item within a TreeView widget. A TreeView is basically nothing but a collection of these types of controls. If you override its CSS class, you can give your TreeItems specific icons and such, in which case you need to change the CSS background-image for those specific MenuItems, which really is out of scope of this article.

Basetypes
Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The (visible) text of your TreeItem.

public string URL

If you give this property a value, the item will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

public bool Expanded

True if this particular TreeItem is actually expanded.

public TreeView TreeView

Will return the parent TreeView of this item. This property will traverse upwards in the ancestor hierarchy and find the first TreeView instance, being an ancestor of this particular item and return that object.

130. Timer

Magix.UX.Widgets.Timer

Description:
Timer widget that will periodically call the server every n'th millisecond. You can set its period through the

Basetypes
Magix.UX.Widgets.Core.BaseControl

Methods

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

protected override string GetClientSideScriptType()

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler Tick
Raised periodically every Duration milliseconds

Properties

public bool Enabled

if true control is enabled and will raise the Tick events every Duration milliseconds, otherwise will not raise tick events before enabled again

public int Interval

Milliseconds bewteen Tick events are raised

131. Window

Magix.UX.Widgets.Window

Description:
Window control. Basically an "advanced panel" with support for moving. It also features borders which can be skinned. The equivalent of a normal desktop window.

Basetypes
Magix.UX.Widgets.Core.CompositeControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler Closed
Raised when window is being closed.

public EventHandler Dragged
Raised when window is moved in browser.

Properties

public string Caption

This is the caption of the header parts of the Window.

public bool Closable

If this property is true, then the window can be closed by clicking a closing icon. The default value is true.

public bool Draggable

If this property is true, then the window can be moved around on the viewport. The default value is true.

132. TreeView

Magix.UX.Widgets.TreeView

Description:
This widget makes it possible for you to create selector kind of widgets that mimicks the representation of a tree kind-of hierarchical structure for displaying things. Useful for displaying for instance folder structures and such.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler SelectedItemChanged
Raised when a new item was selected. Use the SelectedItem property to see which item was actually selected.

Properties

public TreeItem SelectedItem

Returns or sets the currently selected item.

133. TextBox

Magix.UX.Widgets.TextBox

Description:
A single-line type of 'give me some text input' type of widget. This widget is a wrapper around the input type="text" type of widget. If you need multiple lines of input, you should rather use the TextArea widget. However this widget is useful for cases when you need the user to give you one line of text input. See also the RichEdit widget if you need rich formatting of your text. This widget can also be set to 'password mode', which means whatever is typed into the widget will not be visible on the screen. Please notice though that by default, this will be transfered to the server in an unsecure manner, so this is only a mechanism to make sure that other people cannot read over your shoulder to see what you're 'secretly' trying to type into your TextBox. Use SSL or other types of security to actually implement safe transmitting of your passwords and similar 'secret text strings'.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Methods

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Events

public EventHandler EnterPressed
If this event is subscribed to, you will get notified when the carriage return, enter or return key is pressed and released. Useful for being able to give 'simple data' without having to physically click some 'save button' to submit it.

Properties

public TextBoxMode TextMode

The mode your textbox is set to. The default value of this property is Normal which means the characters will display as normal. You can set the mode to Password, which will not show the characters as they're written which may be useful for password text boxes.

public int MaxLength

The maximum number of characters this TextBox allows the user to type in.

134. TabButton

Magix.UX.Widgets.TabButton

Description:
One instance of an item within a TabStrip. Mostly shown as buttons inside of the TabStrip.

Basetypes
Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The caption of your button. Or the visible text.

public string MultiPanelViewID

MultiPanelView ID associated with this instance. Meaning which MultiPanelView will be visible when button is clicked. If this value is not given, the index of the MultiButtonView will be used as the basis of figuring out which view to expand. Meaning that if this button is the 3rd button in the collection, the 3rd view in the view collection will be expanded.

135. SubMenu

Magix.UX.Widgets.SubMenu

Description:
Child control of a MenuItem. Will contain the SubMenuItems of a MenuItem. If you have MenuItems which have children, then you must create an item of type SubMenu and add those children inside of that object. This is true both for .ASPX markup and Menu hierarchies created in code.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

136. TextArea

Magix.UX.Widgets.TextArea

Description:
A multiple line type of 'give me some text input' type of widget. It wraps the textarea HTML element. If you only need single lines of input, you should probably rather use the TextBox widget. However this widget is useful for cases when you need multiple lines of text input. See also the RichEdit widget if you need rich formatting of your text.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

137. TabStrip

Magix.UX.Widgets.TabStrip

Description:
A strip of TabButtons, used often to trigger changing active view in combination with the MultiPanel control. This control is a collection of TabButtons.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler MultiButtonClicked
Server-side Event Handler raised whan any of the buttons are clicked. If you do not handle this event, then the entire logic will run completely on the client-side, which means it'll run significantly faster due to not having to go to the server at all to do its logic.

Properties

public ExpansionTrigger ExpansionMode

What to trigger the opening of the View. Possible values are Click, DblClick and MouseOver.

public int ActiveMultiButtonViewIndex

Currently active or visible view's index

public string MultiPanelID

ID of MultiPanel associated with this instance. This must be set to something since otherwise you have effectively created a completely useless TabStrip, with no purpose.

138. MenuItem

Magix.UX.Widgets.MenuItem

Description:
Items within a Menu. A MenuItem can have a Text property in addition to a SubMenu, which if defined will popup as a child of the MenuItem. The MenuItem class is basically the individual items of your menu.

Basetypes
Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The Caption of the MenuItem. This is the text displayed within your MenuItem.

public string URL

If you give this property a value, the MenuItem will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

public SubMenu SubMenu

Will return the SubMenu of the MenuItem, if any exists. Every MenuItem that have child MenuItems, should create a SubMenu and put its sub MenuItems within the SubMenu portion of their markup, or instantiate them within the SubMenu child control of those MenuItems, if the items are being added programmatically.

public Menu Menu

Will return the parent Menu of the MenuItem. This property will traverse upwards in the ancestor hierarchy and find the first Menu instance, being an ancestor of this particular MenuItem and return that object.

139. CheckBox

Magix.UX.Widgets.CheckBox

Description:
A CheckBox is a 'two state button' which you can turn 'on' and 'off'. Useful for boolean UI situations where use must choose between two options, for instance 'yes' or 'no' situations.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler CheckedChanged
Event raised when the checked state of the widget changes. Use the Checked property to determine if the CheckBox is 'on' or 'off'.

Properties

public bool Checked

Use this property to determine if the widget is checked or not. If this property is true, then the widget is checked. The default value is 'false'. You can also set this value in your code to change the state of the checked value.

140. HiddenField

Magix.UX.Widgets.HiddenField

Description:
Hidden field widget. Useful for having state you wish to pass on to the client but don't want it to be visible for the end user. Notice that this is not a safe place to put things that the user is not supposed to see, like passwords and such. Do NOT trust the value of this element to not be tampered with. Alternatives for using this widget is ViewState, cookies and the Session object. ViewState and cookies are neither 'safe' against tampering.

Basetypes
Magix.UX.Widgets.Core.BaseControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string Value

The value of the hidden field. The default value is string.Empty. Please notice that the HiddenField is not a 'safe' place to store things you do not want to show the end user, so only use this for non-secure-critical information, such as for instance UI states and such. No 'secrets' in this value, since this value is very easy to figure out for the end user.

141. DynamicPanel

Magix.UX.Widgets.DynamicPanel

Description:
Control for making it easier to dynamically instantiate new controls and add them into your page. Completely abstracts away the entire hassle of storing whether or not a control has been previously loaded into the page or not. Make sure you handle the Reload event, and dependent upon the key given will load the exact same control and add it up into the DynamicPanel, and you should experience a very smooth experience in regards to dynamically loading controls into your page. Call the method LoadControl or AppendControl with a unique ID defining which control you wish to load, for instance the name of a UserControl file on disc, and until you explicitly clear your DynamicControl, the same control will be automatically loaded every time. If you do not create a Reload Event Handler for your widget, and you call LoadControl, then the DynamicControl will assume that what you're passing in is a fully qualifed path to a UserControl, and attempt to load it as such. You can use the Extra parameter to add extra initialization parameters into the control upon its first load.

Basetypes
Magix.UX.Widgets.Panel

Methods

public void LoadControl(string key)

Loads a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also AppendControl for having multiple controls within the same DynamicControl. Notice that LoadControl will clear any previously loaded controls from the DynamicControl.

public void LoadControl(string key, object extra)

Loads a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also AppendControl for having multiple controls within the same DynamicControl. The extra parameter will be passed into the Reload event. Notice that LoadControl will clear any previously loaded controls from the DynamicControl.

public void AppendControl(string key)

Appends a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also the LoadControl method.

public void AppendControl(string key, object extra)

Appends a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also the LoadControl method. The extra parameter will be passed into the Reload event.

public void ClearControls()

Clear the controls and makes sure they won't be (re-)loaded again on the next request.

Events

public EventHandler< ReloadEventArgs > Reload
Raised when the DynamicControl for some reasons need to load its control(s). Make sure you load the exact same control every time you're given the same Key value in the ReloadEventArgs parameter of your event handler. Also make sure that the Key is uniquely identifying a control such that you cannot get name clashes.

Properties

public string Key

The entire key for the controls loaded into your DynamicControl. Notice that this might be a list of Keys, separated by pipe (|). Really only exposed for 'advanced functionality'. Do not in general terms use this property for anything. It may be removed in future versions of MUX.

142. AccordionView

Magix.UX.Widgets.AccordionView

Description:
Panels contained inside the Accordion control. Basically an Accordion is only a collection of these types of controls.

Basetypes
Magix.UX.Widgets.Core.CompositeControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Caption

The Header text of the AccordionView. The text displayed at its top.

143. Accordion

Magix.UX.Widgets.Accordion

Description:
Accordions are panels, or AccordionViews grouped together where you can expand and collapse individual AccordionViews within a group. Normally only one AccordionView can be visible at the same time, but this is configurable.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

public void SetActiveView(int index)

Will set the active AccordionView of the Accordion to the specified index.

public void SetActiveView(AccordionView selectedAccordionView)

Sets the active AccordionView to the given one.

Events

public EventHandler ActiveAccordionViewChanged
Event raised when the active AccordionView is changed. If you allow multiple AccordionViews to be opened at the same time, then this event will not be raised.

Properties

public int ActiveAccordionViewIndex

Index of the current Active AccordionView. If you allow multiple AccordionViews to be open at the same time, then this value doesn't make much sense.

public AnimationType AnimationMode

How to animate AccordionViews when opened, see the AnimationType for details.

public ExpansionTrigger ExpansionMode

What to trigger the opening of the View, see the ExpansionTrigger for details.

public int AnimationDuration

How many milliseconds it will take to animate when an accordion is opened. This property doesn't make sense if you're using the Accordion in "NoAnimation" mode.

144. Button

Magix.UX.Widgets.Button

Description:
A clickable button. The equivalent of input type="button". Use when you need a clickable thing to resemble a button. See also the LinkButton for an alternative. Also remember that any Widget in Magix UX can be made clickable, so you can also use a Label as your 'clickable thingie' if you wish. Even though anything can be made clickable in Magix UX, it is often an advantage to use buttons or link buttons since these elements will mostly be recognized by screen readers and such, and it is hence more 'polite' to use these specially designed types of 'clickable objects' such as the Button.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

145. HyperLink

Magix.UX.Widgets.HyperLink

Description:
A wrapper around a hyper link or anchor HTML element (anchor HTML element ...)
Sometimes you will need to create links that might change or needs changes after initially created. For such scenarios, this widget is highly useful.

Basetypes
Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string Text

The anchor text for your hyperlink. This is the text that will be visible in the browser.

public string URL

The URL for your link. This is where the user ends up if he clicks your anchor text.

146. LinkButton

Magix.UX.Widgets.LinkButton

Description:
This widget is another type of 'button widget', though this will be rendered using anchor HTML element (anchor element...) Even though everything can be made 'clickable' in Magix UX, it is definitely semantically much more 'correct' to constraint yourself to the ones that are expected to be 'clickable', such as this widget (LinkButton), Button, ImageButton etc. Among other things screen-readers and such will recognize these types of elements as 'clickable' and present the end user with the option of clicking these types of widgets.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

Properties

public override string Text

The text property of the LinkButton. This is the 'anchor text' of the widget. Basically what the user will 'see' on the screen.

147. Menu

Magix.UX.Widgets.Menu

Description:
Menu control. A Menu is basically a collection of MenuItems. Mimicks a Menu the way you're used to seeing them on desktop systems, or at other web applications. The Magix UX menu have support for any number of child menus, and are very customizable in regards to how it looks and behaves.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler LeafMenuItemClicked
Raised when a Leaf MenuItem, meaning a MenuItem with noe children have been clicked. Normally these are the MenuItems you need to handle on the server, while the ones with children are basically just 'groupings' of other MenuItems.

public EventHandler MenuItemClicked
Raised when any MenuItem have been clicked.

Properties

public ExpansionTrigger ExpansionMode

How to animate items when opened, or what to trigger the opening of the item. Your choises here are Click, DblClick and Hover.

148. Label

Magix.UX.Widgets.Label

Description:
A 'text widget'. The basic purpose of this widget is purely to display text and nothing else, though through the CssClass property and the Style property you can easily manipulate this to do mostly anything you wish. If a more 'complex widget' is needed, for instance to host other widgets, then the Panel widget is more appropriate to use than the Label. Unless the Tag property is changed, this widget will render as a span...

Basetypes
Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string Text

The text property of your label. This is what the user will see of your widget.

public string Tag

The HTML tag element type used to render your widget. You can set this property to anything you wish, including 'address', 'p' or any other types of HTML tags you wish to use to render your widget. If you need a 'div' HTML element though, or you need to render a widget with child widgets, you should rather use the Panel.

public string For

Useful for associating a label with an HTML FORM input element, such as a CheckBox or a RadioButton etc. Notice that this property can only be legally set if the Tag property is of type "label", which is NOT the default value. An exception will be thrown if you attempt at setting this property without changing the Tag property to "span", which is its default value.

149. ImageButton

Magix.UX.Widgets.ImageButton

Description:
Technically this widget is completely redundant towards the Image Ajax Widget. But it is here for cases where you want to be semantically highly correct and need something that gives clues in regards to that it is 'clickable'. If you have an Image which is clickable, then semantically this widget is more correct to use.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string ImageUrl

The URL of where your image is. This is the image that will be displayed to the end user.

public string AlternateText

This is the text that will be displayed if the link to the image is broken. It is also the text that most screen-readers will read up loud to the end user.

150. Image

Magix.UX.Widgets.Image

Description:
Image Ajax Widget. Useful for showing images that needs Ajax functionality somehow. Notice that most times it's more efficient to display other types of widgets, such as the Panel or a Label and set it to display an image through using something such as background-image through CSS or something similar.

Basetypes
Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string ImageUrl

The URL of where your image is. This is the image that will be displayed to the end user.

public string AlternateText

This is the text that will be displayed if the link to the image is broken. It is also the text that most screen-readers will read up loud to the end user.

151. CompositeControl

Magix.UX.Widgets.Core.CompositeControl

Description:
Abstract helper widget for widget developers. Used internally in e.g. Window and Accordion. Useful for those cases where you have a widget which is 'composed' out of both a norml 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself. The Window is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Caption and a Close button. That's why the Window is a 'CompositeControl'.

Basetypes
Magix.UX.Widgets.Panel

Properties

public Panel Content

The Content area of your widget. This is where you would want to add up child controls to make them a part of the child controls collection of your widget.

public Panel Control

This is your actual 'control area'. This is, among other things, where you want to add up Aspects and such if you want to add aspects to widgets which are CompositeControls. In code you can use 'myControl', but in markup you have to use Control to gain access to the control itself. This is because of that the Content property is the default for your control.

152. BaseWebControlFormElementText

Magix.UX.Widgets.Core.BaseWebControlFormElementText

Description:
Abstract base class for WebControls which are BaseWebControlFormElement type of controls, but also have a Text property. Text property is overridable, its default implementation sets the 'Value' property on the client-side, which may or may not be suitable for your needs.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElement

Properties

public string Text

The text that is displayed within the control, default value is string.Empty. This is very often a string next to the widget which the user can read.

153. ViewCollectionControl-g

Magix.UX.Widgets.Core.ViewCollectionControl-g

Description:
Abstract base class for widgets which are supposed to contain only one type of child widgets. Contains a lot of usable shorthands like for instance Enumerable support, and an index operator overload.

Basetypes
Magix.UX.Widgets.Panel

Properties

public IEnumerable< T > Views

Enumerable of all the Views inside the control

public T this[int index]

Shorthand to retrieve a specific View

public T this[string ID]

Shorthand to retrieve the view with the given value as its ID

154. CompositeViewCollectionControl-g

Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Description:
Abstract helper widget for widget developers. Used internally in e.g. TreeItem. Useful for those cases where you have a widget which is 'composed' out of both a norml 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself, and you in addition to this also have a specific type of widgets that are supposed to be its child control collections, like for instance the TreeItem are only supposed to have TreeItems within itself. The TreeItem of the TreeView is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Text property and many other child controls. In addition it is supposed to only have child controls of type TreeItem. That's why the TreeItem is a 'CompositeViewCollectionControl'.

Basetypes
Magix.UX.Widgets.Core.ViewCollectionControl-g

Properties

public Panel Content

The Content area of your widget. This is where you would want to add up child controls to make them a part of the child controls collection of your widget.

public Panel Control

This is your actual 'control area'. This is, among other things, where you want to add up Aspects and such if you want to add aspects to widgets which are CompositeControls. In code you can use 'myControl', but in markup you have to use Control to gain access to the control itself. This is because of that the Content property is the default for your control.

155. BaseWebControl

Magix.UX.Widgets.Core.BaseWebControl

Description:
MUX WebControl equivalent. Contains a couple of additions to the MuxBaseControl. Among others a style attribute and a CssClass. Also contains most of all the DOM event handlers you can possibly handle in Magix UX. In addition to most of the client-side effects you can possibly run on your MUX controls. Class is abstract, and hence cannot be instantiated directly but is meant for inheriting from.

Basetypes
Magix.UX.Widgets.Core.BaseControl

Methods

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

Events

public EventHandler Click
Event raised when user clicks your widget.

public EventHandler DblClick
Event raised when user double-clicks your widget.

public EventHandler MouseDown
Event raised when user pushes his (normally) left mouse button down while the mouse cursor is over your widget. Normally you'd be more interested in handling the Click event. But this event might have its uses too.

public EventHandler MouseUp
Event raised when user have pushed his (normally) left mouse button down while the mouse cursor is over your widget, and then releases the mouse button. Normally you'd be more interested in handling the Click event. But this event might have its uses too.

public EventHandler MouseOver
Event raised when user moves his mouse over your widget. The equivalent of 'hover'.

public EventHandler MouseOut
Event raised when user moves his mouse over your widget, and then moves it away from your widget. The equivalent of 'hover-finished'.

public EventHandler KeyPress
Event raised when a character is typed into the widget. Useful for being able to trap typing into e.g. a TextBox or something similar.

public EventHandler EscKey
Event raised when the escape key is pressed

Properties

public string CssClass

The CSS class(es) of your widget. To explain this is really out of the scope of this library. Pick up any good CSS book to understand how to use this property.

public string Dir

The direction of text within your widget. Legal values are 'rtl' and 'ltr'. These means 'right to left' and 'left to right' and signify the direction of your text. Useful for being able to defined reading directions for non-latin based languages such as Hebrew and Arabic.

public string ToolTip

The tooltip of your widget. Will display a tiny box with your text whenever the user is hovering his mouse over your widget. There are other and richer ways to create more advanced tooltips in Magix UX, but no other ways will demand less bandwidth and less resources on your user's clients. The HTML attribute 'title' is the value actually sent over to the client when the ToolTip is being used.

public Effect ClickEffect

What effect(s) will be ran when the user clicks your widget.

public Effect DblClickEffect

What effect(s) will be ran when the user double clicks your widget.

public Effect MouseOverEffect

What effect(s) will be ran when the user moves his mouse over your widget.

public Effect MouseOutEffect

What effect(s) will be ran when the user moves his mouse out of the surface occupied by your widget on the screen.

public Effect MouseDownEffect

What effect(s) will be ran when the user clicks his left mouse button on top of your widget, but before he releases it.

public Effect MouseUpEffect

What effect(s) will be ran when the user have clicked your widget, and then release his mouse button. In most cases the the ClickEffect will be more appropriate to use, but this event effect has its uses too.

public Effect KeyPressEffect

What effect(s) will be ran when the user have types characters into your widget.

public Effect EscKeyEffect

What effect(s) will be ran when the user clicks ESC

public string TabIndex

Every DOM element on the client-side has a 'tab order' which defines the order widgets will be given focus while the user is tabbing through your web page. The lower this number is, the earlier in the chain the specific widget will be given focus when tabbing. This is the value defining the tab order.

public StyleCollection Style

The inline styles of your widget. An exhaustive explanation of this property is truly out of scope of this file, pick up any good book on the subject CSS to understand how to use this property. But specifically for Magix UX, any style properties added, removed, changed and so on any times during its life-cycle, will automatically, and extremely bandwidth efficient be changed/added/removed automagically by the core engine of MUX.

156. BaseControl

Magix.UX.Widgets.Core.BaseControl

Description:
Abstract base class for mostly all widgets in Magix UX. This is where the Ajax 'core engine' to a large extent exists within Magix UX. Contains several interesting features, such as for instance the Info property, which can take any string information and serialize back and forth between server-requests. If you need to create your own Ajx Control, you should either directly, or most probably indirectly, inherit from this MuxBaseControl. If you inherit, directly or indirectly, form this class you need to override several methods from this class. The most notable is RenderMuxControl. If you have a 'visual widget', you will mostly inherit from MuxBaseWebControl instead of this class. If your control is 'non-visual', such as the MUX Timer, you will mostly inherit from this class.

Methods

protected void RenderJson(HtmlTextWriter writer)

Only override this one if you truly know what you're doing, and you really need some modified logic. See how the Aspects work for a working example of how this method can be overridden.

protected void SetJsonValue(string key, object value)

This one should be used in properties and such for control developers. Please notice that there is no way we can track values being set before the ViewState is finished loading, and hence all properties and such being set before the ViewState is loaded will be ignored...

protected void SetJsonGeneric(string key, string value)

This method will set a 'generic' property for you control. Useful for JS client-side mappings that doesn't really exists since these key/values pairs will be rendered as additional attributes back to the client.

protected void SerializeJSONValue(string key, object value, StringBuilder builder)

Helper method to serialize *ONE* JSON value. Can by default handle a range of different types, such as Rectangle, Point, string, decimal, int, bool etc. If you need to be able to serialize other types, then this can easily be accomplished through overriding this method in your own widget and handle that specific type/property yourself.

protected string GetClientSideScript()

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

protected string GetClientSideScriptType()

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

protected string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

protected void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected string GetControlInvisibleHTML()

Some, but very few, controls need specific in-visible HTML. The default value of this method will be rendering a 'span' with the style value of display:none. Some custom widgets needs to be able to override this markup, for instance because they they needs to render list-elements (HTML li element) to be semantically correct. Override this method to make sure your HTML becomes semantically correct for such cases.

public void ReRender()

Forces the control to re-render as HTML. This is basically a revert to UpdatePanel logic for those extreme cases where such is needed. In general terms you should really only use this method if you either are wrapping non-MUX controls inside of MUX Ajax controls, or you have changed the control collection by either adding or removing new controls from this control's Control collection. There exists several tutorials on this matter at the MUX website.

public void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

public bool AreAncestorsRenderingHTML()

Will return true, if for some reasons, any ancestor control are being rendered as pure HTML. This might happen due to an ancestor control are being re-rendered, or it may happen due to an ancestor control being set to visible for the first time. This is in general terms an *EXPENSIVE* method call since it needs to traverse the entire ancestor hierarchy until it hits the Page. Hence be *CAREFUL* with using it...!!

```
public Dictionary( string, string ) GetJsonValues(string key)
```

Retrieves a specific JSON collection values. Used by the style classes and to add up generic attributes to the DOM elements.

```
public override void Focus()
```

If called, this will set the focus to this widget on the client-side. Please notice that there's no 'multiple calls guard' in this logic, which means if you call this method for several controls, only the last method call will actually 'succeed'...

Properties

```
public   string   Info
```

Additional information you wish to attach to your widget somehow. Useful for things such as Repeaters where you have the same event handler for several buttons on your page, but need to know specifically which button was actually pressed. PS! Just like the HiddenField the value in this property should not be trusted to be securely serialized since internally it uses the ViewState to serialize this additional information, and ViewState is (by default) not encrypted in any ways.

```
public   bool     HasRendered
```

Will return true if this widget has been rendered before. Kind of like an 'IsPostback logic' on a control level. Useful for determining if this is the first time this control is being rendered, or not.

public bool RenderingHtml

Returns true if this control is either not rendered from before, or has been signaled to re-render as HTML (and JS object registration) Notice though that the control still might be rendering HTML due to an ancestor control, any way upwards in this control's ancestor hierarchy has been signaled to re-render. Meaning, this is not by any means any 'absolute definition' of whether or not this control will render HTML. If you need absolute proof for some reasons of whether or not this control will render as HTML you need to check this property, but also the AreAncestorsRenderingHTML method to find out whether or not any of its ancestor controls are re-rendering.

public IEnumerable< AspectBase > Aspects

Iterator for enumerating all aspects within this specific widget.

157. BaseWebControlFormElementInputText

Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Description:
Abstract base class for widgets being BaseWebControlFormElementText type of controls, but also uses the Text portions as an input value which the user can change himself through interacting with the widget.

Basetypes
Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

public void Select()

Selects the whole text portions of the widget and gives the widget focus. Useful for widgets where you know the user will want to change its entire Text content when returned.

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Events

public EventHandler TextChanged
Raised when widget changes its Text property. Normally a TextBox and a TextArea won't raise this event before they're loosing focus, since that's when we know what the Text property is actually changed to.

public EventHandler EscPressed
Raise when widget has focus and user clicks ESC. Useful for being able to create 'discardable operations' types of logic.

158. BaseWebControlFormElement

Magix.UX.Widgets.Core.BaseWebControlFormElement

Description:
Abstract base class for widgets which are HTML FORM type of elements. Abstracts away things such as blur and focus event handling, enabled, keyboard shortcuts and other internals things. Inherit from this one if you intend to create an Ajax Widget which wraps an HTML FORM element.

Basetypes
Magix.UX.Widgets.Core.BaseWebControl

Methods

protected abstract void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler Blur
Event being raised when the widget does no longer have focus. The exact opposite of the Focused event. Will trigger whenever the widget loses focus for some reasons. Notice that the widget (obviously) needs to *have* focus before it can 'lose' it. Some ways a widget may lose focus is when the user tabs through the widgets on the screen. Another way is clicking another widget, and thereby loosing focus for the previously focused widget.

public EventHandler Focused
The Focus event. The exact opposite of the Blur event. This one will be raised whenever the widget gains focus. Gaining focus can happen several ways, first of all the user might click the specific widget, which will give that widget focus. Secondly the user might tab through the widgets until the specific widget is reached. When the widget gains focus, this event will be raised.

Properties

public string AccessKey

What keyboard shortcut the user will have to use to mimick a 'click' on the widget. Often the keyboard shortcut will be mixed up with other keys, depending upon your operating system or browser. For FireFox on Windows for instance the keyboard combination is ALT+SHIFT+whatever key you choose here. So if you choose 'J' as the AccessKey, the user will have to hold down ALT+SHIFT and press 'J' to use the keyboard shortcut.

public bool Enabled

Boolean value indicating whether or not the widget is enabled. If this one is false, then the user cannot change the value of the widget by interacting normally with it. Please note that this is not safe and can be overridden by tools and manipulating the HTTP stream directly and so on. Do not trust values from controls that are disabled to be the value you explicitly gave them earlier.

public Effect BlurEffect

What Effect will run when user is making the widget lose focus.

public Effect FocusedEffect

What Effect will run when user is making the widget gain focus.