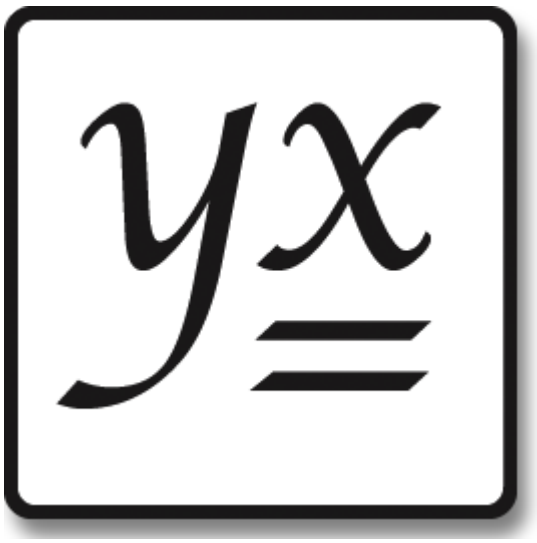


Absyntax Framework

User Guide
December 2013

Table of Contents

Introduction

What Absyntax Is

Absyntax is a desktop framework application whose purpose is to bridge the gulf between people who need bespoke software and people who can deliver such software. You may think of it as a tool for creating software without the need to understand programming languages and jargon usually associated with the discipline of software engineering.

Absyntax is suited to creating both standalone software (i.e. software that is designed to run on its own) and partial software (i.e. software that is designed to be invoked by other computer applications).

What Absyntax Is Not

Absyntax is not a code-generating application. Neither is it a workflow application.

Why Might You Want It?

Absyntax enables a wide range of people – not just expensive specialists – to effect software change. This means that the dependency of an individual or an organisation on traditional software engineering skills is reduced. In turn, this allows the creation and modification of software to be realised more quickly and more cheaply.

Absyntax can be integrated with third-party applications, enabling these applications to invoke Absyntax projects programmatically. This is of particular benefit to packaged solutions that cannot fully meet all of their various clients' needs. By working in tandem with Absyntax, such solutions can support client-specific functionality while avoiding the need for multiple product versions. Even modular solutions with parameter-driven operations are, ultimately, limited. With Absyntax, though, such limitations can be bypassed, providing clients with the capabilities they would expect from bespoke solutions.

Third-party applications require periodic modification, and not only to introduce enhancements or fix bugs. Changing business requirements, legislative impacts and improved analytical methods are just a few of the reasons why further software development might be needed. Some software-driven operations are simply more volatile than others: Absyntax can be used to handle such operations, affording business analysts and other industry specialists the opportunity to effect changes. This softens the impact of such changes on your organisation and offers a more responsive, cost-effective service to your clients.

Many software solutions support an application programming interface (API) allowing software developers to tap into the capabilities of these solutions using programmatic means. But why should such capabilities be accessible only to those with software engineering skills? Absyntax makes it possible to expose APIs to a much wider audience.

Absyntax allows software to be visualised. Discrete, low-level operations can be aggregated for presentational purposes, so you don't need to produce diagrams separately using some other tool.

Absyntax is extensible. This means that software developers can author their own features, which is of particular benefit when needing to provide higher-level reusable operations to a specific industry. Such custom features typically obviate the need for clients to create equivalent

functionality using perhaps dozens of out-of-the-box Absyntax features. Also, developers can integrate existing software components with Absyntax using very little code. This means that Absyntax can harness the power of your existing code libraries without the need for major redevelopment.

Every Absyntax project has a signature. A signature is a function of both a project's state and the underlying codebase. Thus project signatures may be used in support of corporate governance in situations where business-critical operations rely on the execution of Absyntax projects. If a project is changed in some material way or if the underlying framework software changes, the project's signature will change. So if behavioural assurance is important to you then implement your critical operations using Absyntax.

Fundamentals

This section introduces the core aspects of the Absyntax framework. If you are new to Absyntax then this is the place to start.

Projects

An Absyntax project is equivalent to a program: it represents Absyntax's "unit of execution". When you want to create software with Absyntax, you start with an empty project.

Every project supports:

- an entry point that is signalled when project execution begins;
- an exit point that the project signals in order to end.

A project is also a feature group, meaning that it contains features. It is these contained features – and their interconnections – that define the behaviour of a project and thus the behaviour of your software. They may be connected to a project's internal entry- and exit-point proxies, allowing them both to respond when the project receives inputs from the outside world and to signal to the outside world that an operation has concluded.

Projects may be saved as files.

Feature Groups

Perhaps unsurprisingly, a feature group is a group of (or container for) other features. Importantly, a feature group is itself a feature. This means that a feature group has all the attributes of a feature: for example it can be named, and it supports connectors.

Feature groups are very useful for things such as:

- creating custom features through feature aggregation;
- performing well-defined operations whose behaviour is determined by the contained features;
- creating hierarchical projects that compartmentalise discrete operations, foster ease of maintenance and promote understanding.

They are essential for enabling concurrent processing on multiple computers.

In Absyntax the common implementation of the feature group is known as the "configurable feature group". A configurable feature group is a feature group whose inputs and outputs are user-definable (which contrasts with the majority of features, whose connector profiles are fixed). Not all groups are configurable, though.

Features

Features are discrete units of functionality. In other words, features do things. They are the cogs in the machine. They also communicate with one another via inputs and outputs.

Features are the proverbial "black boxes": *how* they do what they do is unimportant (although the ways in which they respond to inputs and trigger outputs *are* important because such details will have a bearing on how you connect your features). As is the case for integrated circuits in the electronics industry, a feature's designer doesn't care about the context in which the feature is used: the designer merely states the feature's behaviour and it is up to the user to determine whether and how to use the feature.

All features have names, which you may change. Within a group, all feature names must be unique.

Connectors

A given feature typically has one or more inputs and outputs (which are also referred to as *plugs* and *sockets* respectively, and collectively known as *connectors*). In general, inputs are used to set a feature's state or initiate a feature's behaviour; outputs are used by a feature to emit signals and data (often in response to the receipt of input signals and data). Connections are created between pairs of connectors in order to pass signals or data from one feature to others.

Each and every input and output can transmit either signals or data. Precisely what is transmitted is determined by the feature's designer and is immutable (i.e. a signal output will always transmit signals; a data input will only ever accept data of a certain type). An input may be connected to multiple outputs. Likewise, an output may be connected to multiple inputs. Inputs and outputs may be left unconnected. Inputs and outputs belonging to the same feature may be connected to one another. However, there are overarching rules that determine whether an input and output may be connected.

All connectors have names, which you may change. Within a feature, all such names must be unique.

Signals and Data

Signals are like empty envelopes. They can be sent and received but carry no information. A signal merely informs its recipient that something has happened. For example, consider the **Collection** feature, which allows you to examine each item in a collection sequentially. When the end of the collection is reached, the feature emits a signal via its EndOfSet output. If you have a feature that should be activated in some way when this happens, you would connect it to the **Collection**'s EndOfSet output.

Data, on the other hand, can be thought of as signals with payloads. Or like envelopes that contain letters. Each letter is an item of information. Take the **ItemList** feature: if you want to add an item to the list, you would connect its Add input to an output that emits data of a compatible type.

Connections

It is unlikely that your project requirements will be met by just one feature. Normally you will need to combine the behaviours of several features in order to realise the functionality you need, something that is achieved by connecting inputs and outputs together. The rules for connecting inputs and outputs are as follows:

- signal outputs may be connected to signal inputs;
- signal outputs may *not* be connected to data inputs;
- data outputs may be connected to signal inputs;
- data outputs may be connected to data inputs *as long as the data types are compatible.*

"Compatible", here, means that the type of the output data is either the same as the type of the input data or is convertible to the type of the input data. See the section on data conversion for more information.

When a feature emits a signal or data through an output, each of the connected inputs must be informed. Depending on how the output has been configured, the connected inputs are informed either in sequence (i.e. *synchronously*) or at the same time (i.e. *asynchronously*). If they are informed in sequence, the order is determined by a unique number assigned to each of the output's connections. You can change these numbers to control the order in which connections are fired. If the inputs are informed at the same time then no guarantee can be made as to which inputs will be informed first. You must decide on the appropriate behaviour and add synchronisation mechanisms to your projects where necessary.

Data Types

Definition

Data: that's another word for information. But what is a data *type*? And why should you care?

Objects and Types

For several decades, programmers have been developing software based on the concept of *objects*. Put simply, an object is something that has characteristics and can do things. For example, a car is a kind of object: a typical car has an engine capacity and a colour. It also starts when the ignition is turned on. In reality, cars – indeed, any types of object – have all sorts of characteristics and behaviours but we will usually only be interested in a few of them. In the world of Absyntax, an object is an item of data.

Programmers define *types*. A type is like a template: it's a complete definition of the characteristics and behaviours that all of its objects are to support. Thus when we have an object of type "Car", we know that we will be able to find out its engine capacity and its colour, and we also know that we can make it "start" (whatever that might mean). Absyntax features define their data inputs and data outputs in terms of the types of data they support. Thus if a particular input has a data type of “Car" then it will only accept items of data that are cars.

Object Composition

Let's take this a step further. Engine capacity is probably most usefully expressed as a whole number of cubic centimetres. But in the world of software, a whole number is also a type of object. And what about colour? Perhaps text could be used to describe a car's colour, but one man's "purple" might be another man's "indigo". So perhaps colour would be best defined in terms of its RGB content. In other words, we could define a type called "Colour" with the characteristics "R(ed)", "G(reen)" and "B(lue)", each of which could be a whole number in the range 0 to 255. In which case "indigo" could be represented using values for R, G and B of 75, 0 and 130 respectively.

What has just been described is *object composition*. It means that objects can be expressed in terms of other objects. A car, for example, can be expressed in terms of a number (engine capacity) and a colour. In turn, colours can be expressed in terms of three numbers. This is useful to know because it will help you to:

- select suitable data types when necessary;
- understand how information in Absyntax is managed and presented;
- connect inputs and outputs correctly;
- decompose objects into the information you need;
- analyse how your projects are working.

It is also useful to understand *inheritance* because this will help you to understand type conversion. Consider that the "Car" type might have characteristics in common with other types. Hair dryers, for example, also have colour and can be "started". That said, they don't normally have engine capacities. Programmers often recognise such scenarios as this and define their templates accordingly. Both the "Car" type and the "HairDryer" type share some details, and so a programmer might define a type named "PoweredItem", which supports a colour and can be started. So, if the "Car" type is defined as being a type of "PoweredItem" then it needs only to define engine capacity as an additional characteristic because it inherits the characteristics of its

parent type. This means that, say, a data input supporting "PoweredItem" types will accept both items of type "Car" and items of type "HairDryer".

Namespaces

Commonly, the name assigned to a type is qualified by a namespace. An example is "MI2.Units.Lengths.Metre", which includes the namespace ("MI2.Units.Lengths") and the type name ("Metre"). A namespace is a naming convenience used by type designers for the purpose of organising types within a software library or application. The Absyntax Editor displays namespace-qualified type names in several places but is generally of academic interest to Absyntax users and mentioned here for information only.

Data Conversion

In order to transfer data from a feature output to a feature input, a connection must first be established. For a connection to a data input, the reciprocal output data type must be compatible.

Absyntax supports several ways of converting data from one type into another and it will hide any inherent complexity from you. In general, though, the chance of any given data type being convertible to any other type are low. This is because there is a potentially unlimited number of data types that we might define. What do a tree and a tennis ball have in common, and how might we convert from one to another? This depends on how these types have been defined and what, if any, conversion methods have been made available to Absyntax behind the scenes. The good news is that you do not need to know these details. Absyntax will only allow you to create connections that make sense.

Revisiting the previous example concerning cars and hair dryers, we established that both are types of powered item. So whenever we have a car we also (implicitly) have a powered item. This means that a "Car" output could be connected to a "PoweredItem" input. But when we consider this in reverse, it is not possible to say that if we have a powered item it must be a car: it could be a hair dryer. This means that a "PoweredItem" output could not, in general, be connected to a "Car" input.

There is a middle ground. By way of example, consider "46". You have probably assumed that this is a number. But in the context of this document, "46" is no different to any of the other text you are reading. Importantly, your computer does not store it as a number and so cannot do number-related things with it (like add it to another number).

It is sometimes the case that data of one type can be converted to data of another type as long as it is in the right format. In the case of text data, "46" can be converted to the number 46 whereas "46a" cannot. Absyntax cannot know in advance whether an item of text will be in a format that is suitable for conversion to a number, but it can optionally allow you to create connections requiring these kinds of conversion. It does so on the understanding that you know what you're doing and accept that the transmission of data from output to input is not guaranteed because the conversion may fail.

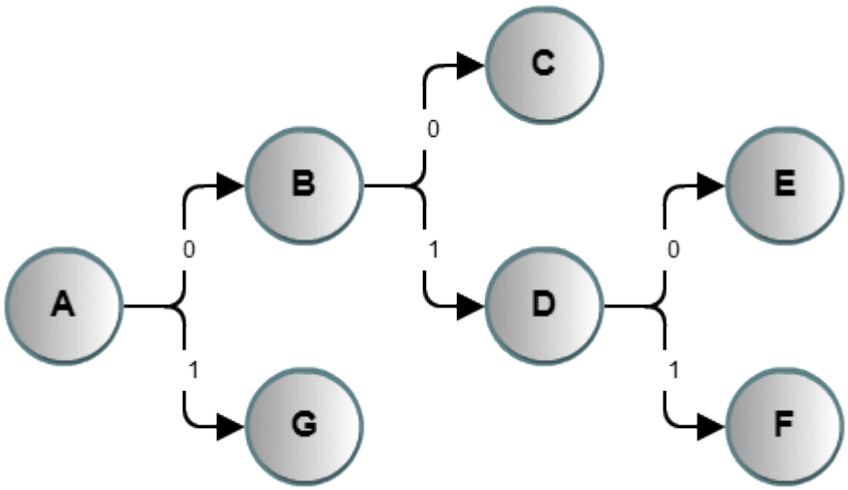
Synchronisation

Threads

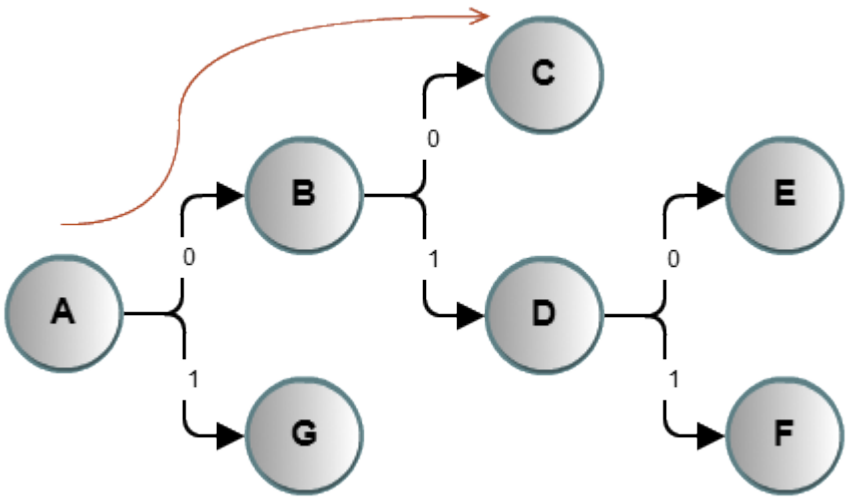
Threading is prevalent in computing although, as a user of computer applications, you have no need to be aware of what threads are and what they can do for you. However, Absyntax cannot mask the concept of threading. This is because the way your project manages and handles threads will have a fundamental bearing on its behaviour.

Think of a thread as a worm of activity: its tail is fixed at its point of origin and it extends itself along connections and through features. When your project starts running, a thread enters through the project's Entry input. Then, depending on how you have configured your entry point, the thread will either be routed along each connection in turn or it will spawn a new thread for each connection. Thereafter, the same rule applies: whenever a thread emerges from a feature it will either be routed along each connection in turn or it will spawn a new thread for each connection, depending on how the output has been configured. Finally, when a thread has no more connections to serve, it disappears.

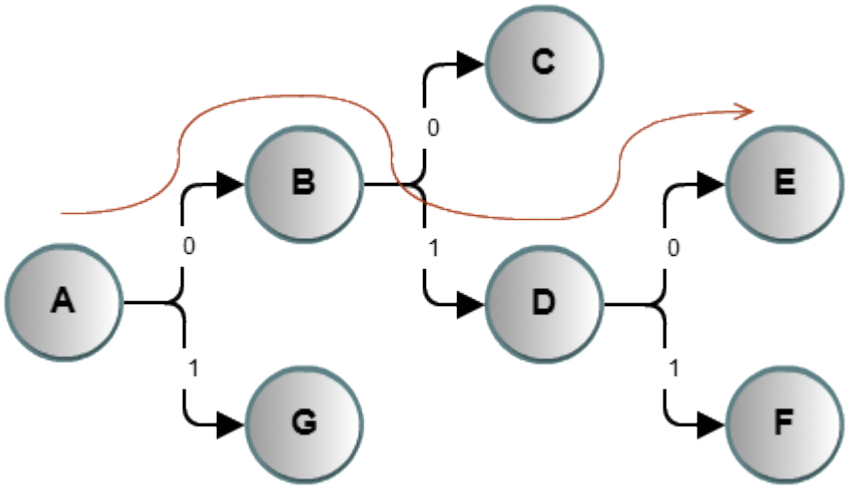
If the same thread is required to serve two connections in turn (i.e. *synchronously*), it will only serve the second connection once it has finished with the first. To understand what this means in practice, consider the following diagram. This illustrates seven features, labelled A through G, and the connections between them. The connection numbers indicate the order in which they are to be served by a single thread. Suppose that all outputs are configured to operate synchronously.



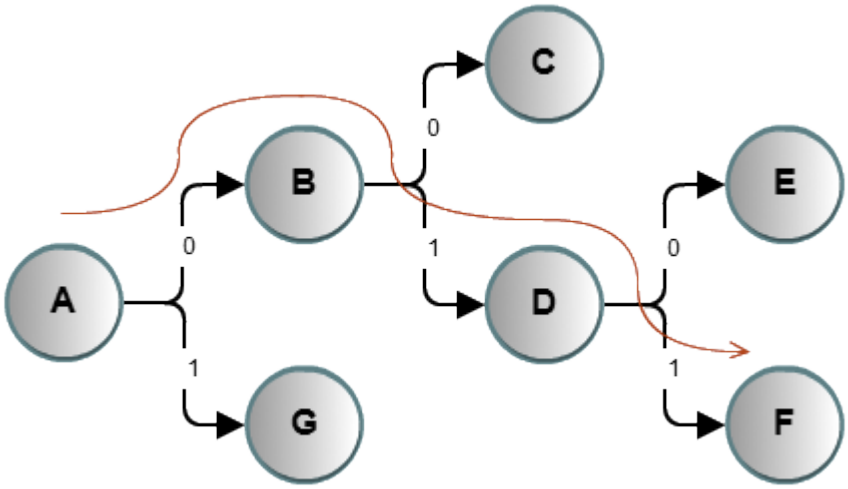
A thread passing out of feature A is required to serve connections to features B and G. Firstly it visits B, which does some work with it before dispatching it to C. Feature C does not trigger any outputs and so, once it has finished with the thread, the thread is free to serve the next connection of B's output.



Next, the thread visits feature D which does some work with it before dispatching it to E.



Feature E does not trigger any outputs and so, once it has finished with the thread, the thread is free to serve the next connection of D's output, which takes the thread on to feature F.



Feature F does not trigger any outputs, so finally the thread winds its way back to the next connection of A's output, which takes it on to feature G. You will need to rely on the descriptions and documentation for a feature and its inputs in order to determine the circumstances in which a particular output is triggered.

Re-entrant Paths

As a single thread extends itself by passing from feature to feature, your computer will set aside more and more memory. If a thread is allowed to continue unchecked, there will come a time when your computer runs out of resources and abruptly terminates your project. The point at which this happens depends largely on the amount of memory your computer has, but it is generally not a concern unless your project has synchronous connection paths that loop back on themselves. Such paths are referred to as *re-entrant paths*.

The good news is that this lurking threat can always be mitigated through the judicious use of feature outputs configured for asynchronous operation. The Absyntax Editor will warn you when it detects re-entrant paths. Note that Absyntax cannot know whether an input to any given feature will trigger an output. Furthermore, if a re-entrant path is only executed a few times then it is unlikely to cause problems. Thus the existence of a re-entrant path is not necessarily cause for concern. Nonetheless it is considered good practice to eliminate these paths from your projects where practicable.

Parallel Computing

As well as facilitating asynchronous communication between features, Absyntax makes it easy to perform truly parallel activities. It achieves this through use of a feature dedicated to the task of managing a group of other features operating in another process, typically on another computer. This feature, **RemoteFeatureGroup**, works in conjunction with the Runtime Server, which is Absyntax's runtime hosting application.

There is no limit to the number of **RemoteFeatureGroup**s you can include in a project.

System Requirements

Absyntax is a suite of Microsoft .NET Framework-based libraries and applications.

Operating Systems

- Windows XP (SP3) (x86)
- Windows Vista (SP2) (x86 and x64) (recommended)
- Windows 7 (SP1) (x86 and x64) (recommended)
- Windows 8 (x86 and x64) (recommended)

Software Requirements

Microsoft .NET 4.0 (installed automatically with the product if necessary, subject to your agreement).

Hardware Requirements (guideline only)

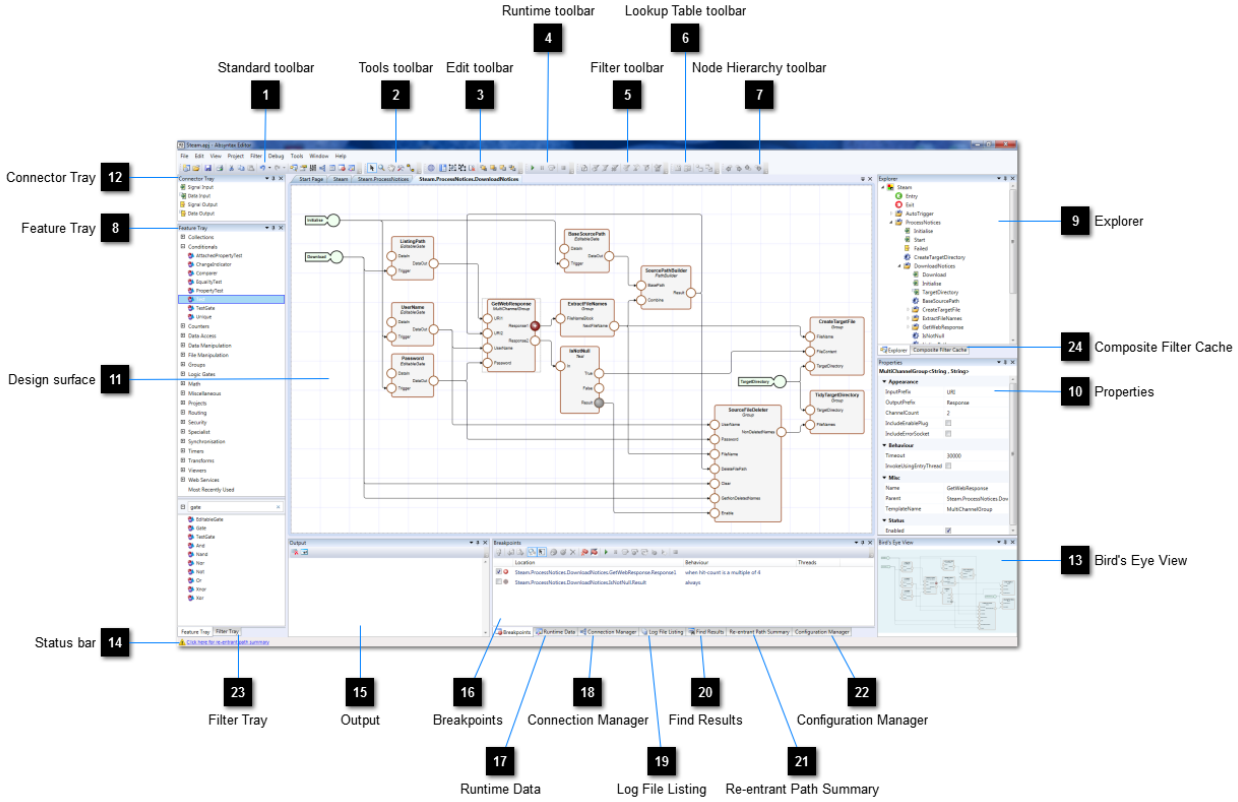
- 2GHz processor
- 512MB RAM
- 1GB (32-bit) or 2GB (64-bit) available hard disk space for Microsoft .NET 4.0
- 10MB available hard disk space for the Absyntax Framework

Installation and Licence Activation

1. Download the zip file containing the latest product version from the Absyntax website.
2. Unzip the contained files: setup.exe and AbsyntaxSetup.msi.
3. Double-click setup.exe to begin the installation.
4. Follow the step-by-step guide. If prompted by a User Account Control dialogue, answer Yes.
5. Once Absyntax has been installed you will firstly need to activate a licence, so start the Absyntax Licence Manager. A desktop shortcut should have been created for this. Alternatively, from the **Start** menu select **All Programs → MII Ltd → Absyntax Licence Manager** or open the Absyntax framework's installation directory and run AbsyntaxLicenceManager.exe. Note: *you must be connected to the internet in order to activate licences*.
6. If you have purchased a licence then enter the purchased licence key and the name of the licensee you specified during the purchase process; then click Add.
7. Activate your purchased licence or, if you do not have one, activate the trial licence that was included with the installation.

See the Licensing Manager section for more details on licence key registration and activation.

Editor



The Absyntax Editor is the framework's primary application for creating, debugging and maintaining Absyntax projects. The main window, shown above, is a container for panels and tabbed documents which may be docked or floated to suit your particular workflow and screen configuration.

1 Standard toolbar

The Standard toolbar provides convenient access to the most common operations and actions.

2 Tools toolbar

The Tools toolbar is used to determine the currently selected design surface tool.

3 **Edit toolbar**

The Edit toolbar provides convenient access to common design surface operations and actions.

4 **Runtime toolbar**

The Runtime toolbar is used to execute and debug the current project interactively.

5 **Filter toolbar**

The Filter toolbar is used to manage Filter Builder operations, typically on behalf of features and other components that make use of Boolean expressions.

6 **Lookup Table toolbar**

The Lookup Table toolbar provides convenient access to common Lookup Table Editor operations and actions, typically on behalf of features that consume lookup tables.

7 **Node Hierarchy toolbar**

The Node Hierarchy toolbar facilitates the positioning of hierarchical components in contexts that support such operations.

8 **Feature Tray**

The Feature Tray contains the templates from which all of a project's features are created.

9 **Explorer**

The Explorer summarises the contents of your project and offers a means of navigating it.

10 **Properties**

The Properties panel displays the state of the selected object or objects.

11 **Design surface**

The design surface of a view is where you add, configure and connect your project's features. Most of your project development work will utilise such views.

12 Connector Tray

The Connector Tray contains templates of connectors that you use to define the inputs and outputs of configurable feature groups.

13 Bird's Eye View

The Bird's Eye View displays the entire content of the currently active view. Positioned on top is a semi-opaque thumbnail representing the extent of the current viewport. Drag the thumbnail around to change the position of the viewport relative to the entire content.

This panel is useful if your view contains a lot of features and you are zoomed into a relatively small area of the view. To show it, select **View → Bird's Eye View** from the menu bar.

14 Status bar

Warns of issues that you should be aware of that may prevent your project from operating correctly.

15 Output

The Output panel displays textual information during project execution.

16 Breakpoints

The Breakpoints panel allows you to manage the way in which threads are interrupted during project execution.

17 Runtime Data

The Runtime Data panel allows you to explore your project's data during execution.

18 Connection Manager

The Connection Manager allows you to visualise and maintain connections between features' inputs and outputs.

19 Log File Listing

The Log File Listing provides easy access to Absyntax's runtime log files.

20 Find Results

The Find Results panel is shown automatically every time you perform a find operation.

21 Re-entrant Path Summary

The Re-entrant Path Summary highlights feature interconnections that may cause your project to fail.

22 Configuration Manager

The Configuration Manager highlights services that your projects requires that require additional information in order to operate correctly.

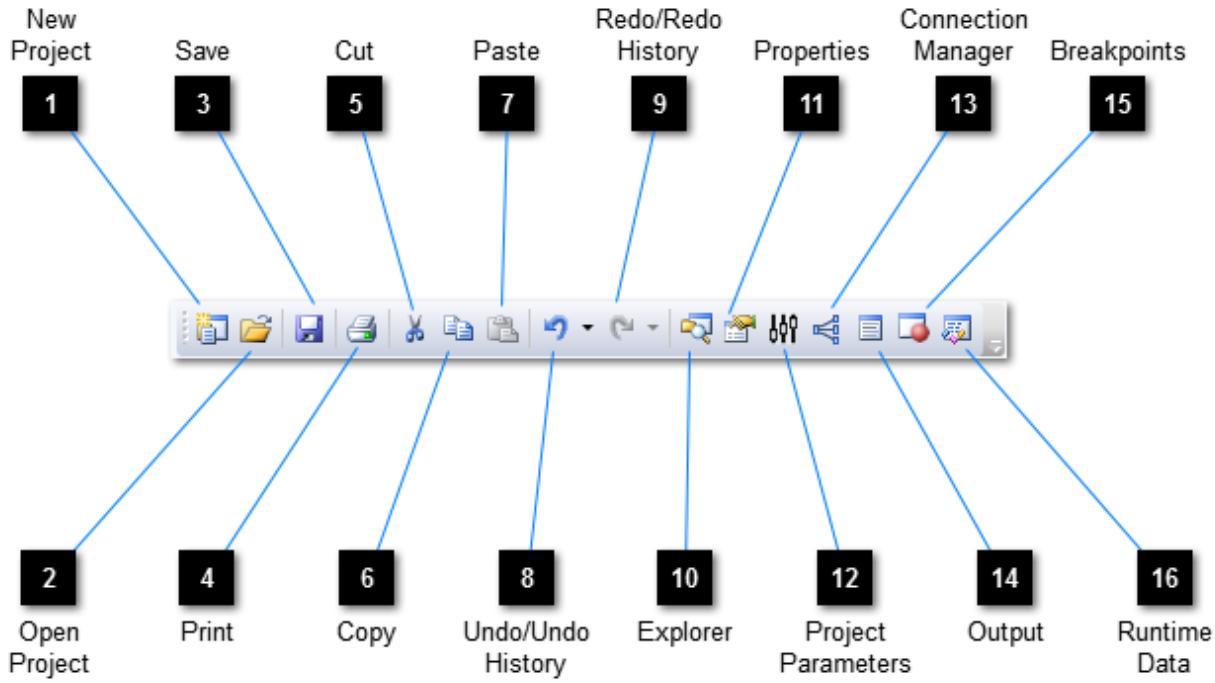
23 Filter Tray

The Filter Tray contains templates for filter elements and groups, which may combined in the Filter Builder to create composite filters.

24 Composite Filter Cache

The Composite Filter Cache contains your saved composite filters.

Standard toolbar



You can show or hide this toolbar by selecting **View** → **Toolbars** → **Standard** from the menu bar.

1 New Project

Discards the current project and displays the Select Project Type dialogue in readiness for creating a new project. If the existing project has not been saved, you will be prompted to do so before continuing.

2 Open Project

Discards the current project and displays the Open dialogue, allowing you to load a previously saved project from file. If the existing project has not been saved, you will be prompted to do so before continuing.

3 Save

Saves the current project to file. If the project is a new project that has not previously been saved then you will be prompted to specify a file name, which defaults to the project name.

If the active document is a lookup table, this will be saved to file instead. If the lookup table is a new lookup table that has not previously been saved then you will be prompted to specify a file name, which defaults to the lookup table's name.

4 **Print**

 Prints the contents of the active document.

5 **Cut**

 Cuts the currently selected item to the clipboard.

6 **Copy**

 Copies the currently selected item to the clipboard.

7 **Paste**

 Pastes the content of the clipboard.

8 **Undo/Undo History**

 Undoes the most recent action. Click the drop-down arrow to reveal a list of all undoable actions and select those that are to be undone.

9 **Redo/Redo History**

 Redoes the most recently undone action. Click the drop-down arrow to reveal a list of all redoable actions and select those that are to be redone.

10 **Explorer**

 Shows the Explorer panel.

11 **Properties**

 Shows the Properties panel.

12 **Project Parameters**

 Shows the Project Parameters panel.

13 **Connection Manager**

 Shows the Connection Manager panel.

14 **Output**

 Shows the Output panel.

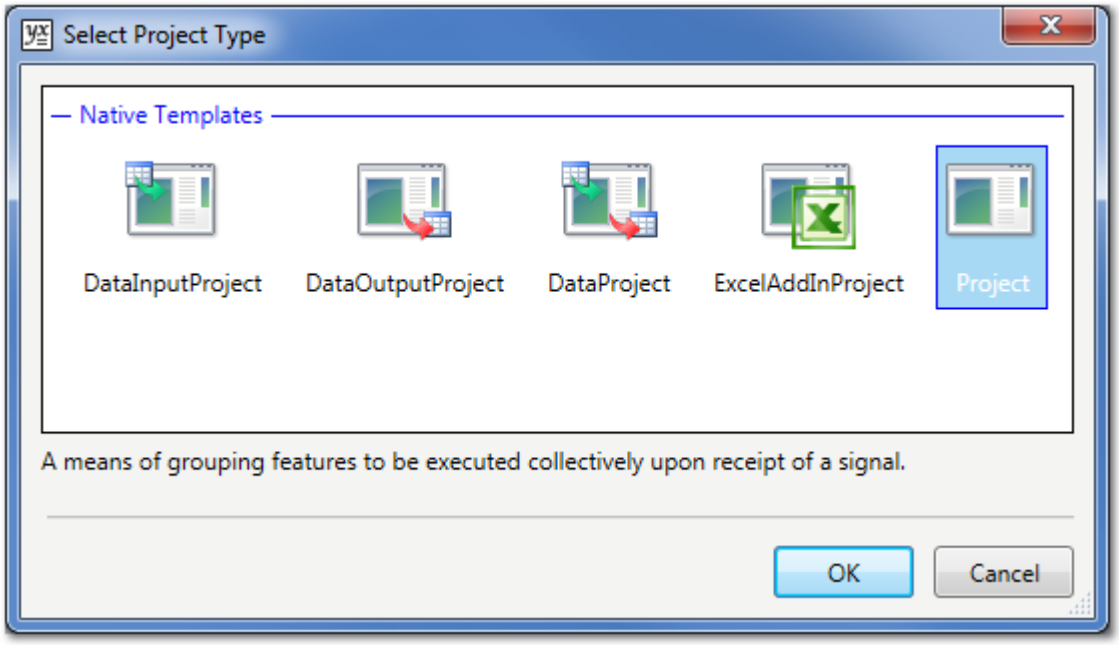
15 **Breakpoints**

Shows the Breakpoints panel.

16 **Runtime Data**

Shows the Runtime Data panel.

Select Project Type



This dialogue is opened whenever you indicate that either:

- a new project is to be created, or
- the type of the current project is to be changed.

Select a project template from the list of known templates and press OK to confirm the selection. If you select a template for a project that either receives or emits data of a type or types that have not been predetermined by the template, the Type Browser will be shown to allow you to specify one or both data types. If you are unsure which template to use, select Project (as shown above). You can always change the project type later by selecting **Project → Change Project Type...** from the menu bar.

Saved project file extensions

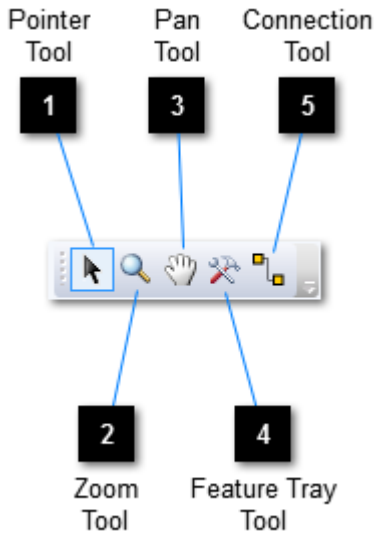
The Editor saves and loads project files with .apj extensions. Such files contain proprietary binary representations of the operational aspects of Absyntax projects. These files are also used by the Batch Client.

When the Editor saves a project, it also saves the project workspace in a separate file with a .apw extension. A project workspace contains details such as project settings, the location of features on the design surface and the shape points of connections between connectors. Workspace files are only relevant to the Editor.

The Editor can load a .apj file without an accompanying .apw file but the project's content will not be presented as its creator intended. If you come across this situation, use the Auto Layout facility.

In general, though, if you are sharing projects then it is good practice to keep .apj and .apw file-pairs together.

Tools toolbar



You can show or hide this toolbar by selecting **View** → **Toolbars** → **Tools** from the menu bar.

1 Pointer Tool

 Use this tool to perform a variety of miscellaneous design surface tasks, including selection and repositioning. Use the mouse wheel to scroll up and down.

The following keyboard modifiers apply to the Pointer Tool:

Shift	The mouse wheel will scroll left and right
Ctrl	The Pointer Tool becomes the Zoom-in Tool
Space Bar	The Pointer Tool becomes the Pan Tool
Alt	The Pointer Tool becomes the Connection Tool

2 Zoom Tool

 Use this tool to zoom in to and out of the design surface. Click to zoom in by preset increments. Click and drag to create a zoom rectangle defining the extent of a zoom-in operation. Use the mouse wheel for continuous zoom-in/out.

The following keyboard modifiers apply to the Zoom Tool:

Alt	All click/drag operations become zoom-out operations
Space Bar	The Zoom Tool becomes the Pan Tool

3 Pan Tool

 Use this tool to drag the design surface around within the viewport. The Pan Tool is only of use when the design surface extends beyond the bounds of the viewport.

The following keyboard modifiers apply to the Pan Tool:

Shift	The mouse wheel will scroll left and right

4 Feature Tray Tool

 Use this tool to add new features to the design surface.

The following keyboard modifiers apply to the Feature Tray Tool:

Ctrl	The Feature Tray Tool becomes the Pointer Tool

5 Connection Tool

 Use this tool to create connections between pairs of connectors. The rules for creating connections are covered here. Click down on any connector and drag the connection outline to another connector. While hovering over another connector, you will see one of four cursors.

 You are trying to connect an input to an input or an output to an output, which is not possible.

 The attempted connection would contravene the connection rules.

 The connection is allowed but there may be data conversion problems at runtime.

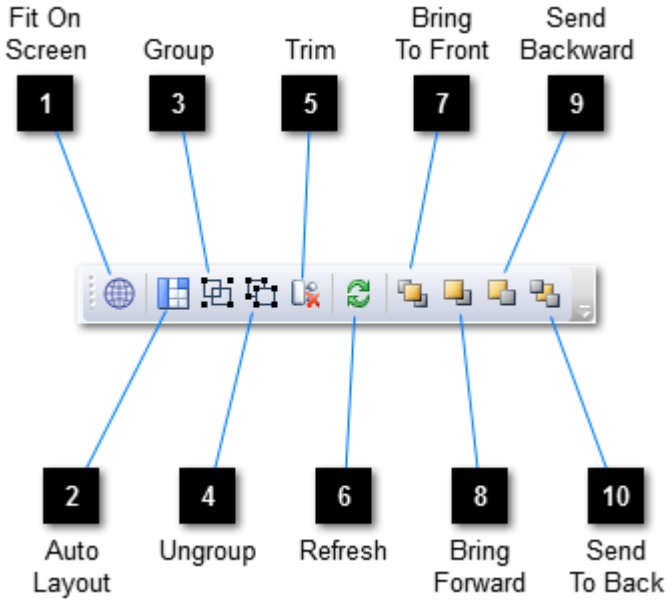
 The connection is allowed and there will be no data conversion problems.

The following keyboard modifiers apply to the Connection Tool:

Shift	The mouse wheel will scroll left and right
Ctrl	The Connection Tool becomes the Zoom-in Tool
Space Bar	The Connection Tool becomes the Pan Tool
Alt	The Connection Tool becomes the Pointer Tool

The Connections Manager offers an alternative means of creating connections.

Edit toolbar



You can show or hide this toolbar by selecting **View** → **Toolbars** → **Edit** from the menu bar.

1 **Fit On Screen**

Fits the contents of the active document into the viewport.

2 **Auto Layout**

Automatically positions all features in the active document.

3 **Group**

Groups the selected features using a standard, connector-configurable **Group** feature.

4 **Ungroup**

Ungroups the features of any groups in the current selection.

5 **Trim**

Removes any redundant connectors from the selected, connector-configurable groups.

6 **Refresh**

 Refreshes the currently loaded project by identifying and refreshing all refreshable features therein. Note that this action is not undoable.

7 **Bring To Front**

 Brings the selected items to the top of the z-order. Note that features and connections are rendered in different layers and it is possible to position either of these layers uppermost by setting the rendering intent accordingly.

8 **Bring Forward**

 Brings the selected items forward in the z-order. Note that features and connections are rendered in different layers and it is possible to position either of these layers uppermost by setting the rendering intent accordingly.

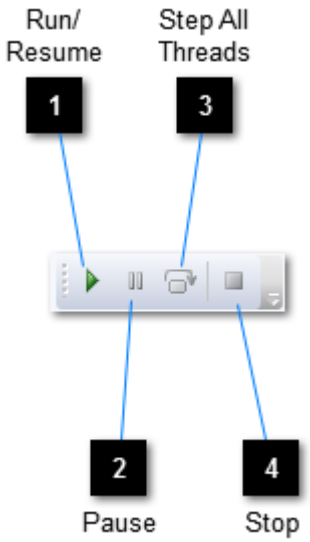
9 **Send Backward**

 Sends the selected items backward in the z-order. Note that features and connections are rendered in different layers and it is possible to position either of these layers uppermost by setting the rendering intent accordingly.

10 **Send To Back**

 Sends the selected items to the bottom of the z-order. Note that features and connections are rendered in different layers and it is possible to position either of these layers uppermost by setting the rendering intent accordingly.

Runtime toolbar



You can show or hide this toolbar by selecting **View** → **Toolbars** → **Runtime** from the menu bar.

1 Run/ Resume

Starts executing the current project or resumes all of a project's paused threads.

2 Pause

Pauses all threads in the currently executing project.

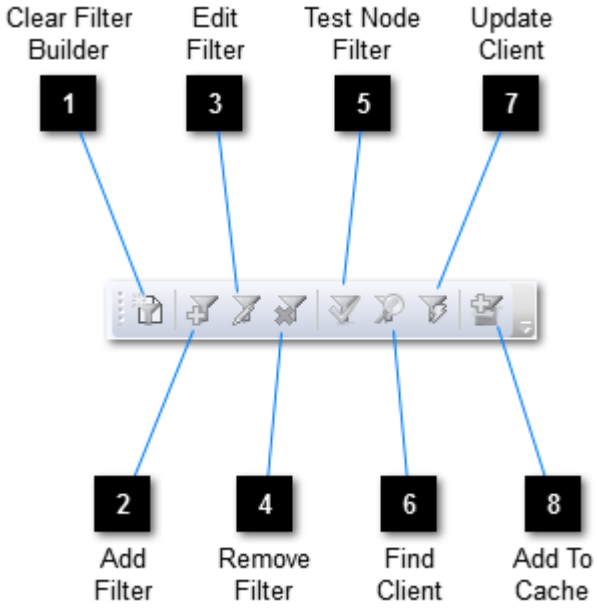
3 Step All Threads

Steps all paused threads in the currently executing project.

4 Stop

Aborts the currently executing project.

Filter toolbar



The Filter toolbar works in conjunction with the Filter Builder and with components that make use of Boolean expressions. You can show or hide this toolbar by selecting **View → Toolbars → Filter** from the menu bar.

1 Clear Filter Builder

Clears the Filter Builder of its current composite filter.

2 Add Filter

Adds a filter to the selected component by firstly clearing and opening the Filter Builder. The Filter Builder then considers the component to be its "client" (see below). The composite filter that you create is only added to the client when you choose Update Client.

3 Edit Filter

Edits the selected component's existing filter by firstly opening the Filter Builder and setting its composite filter. The Filter Builder then considers the component to be its "client" (see below). Any changes you make to the composite filter are only passed on to the client when you choose Update Client.

4 Remove Filter

Removes the existing filter from the selected component.

5 **Test Node Filter**

 Opens the Test Filter dialogue in respect of the currently selected Filter Builder node.

6 **Find Client**

 Opens the view containing the Filter Builder's current client component and highlights said component.

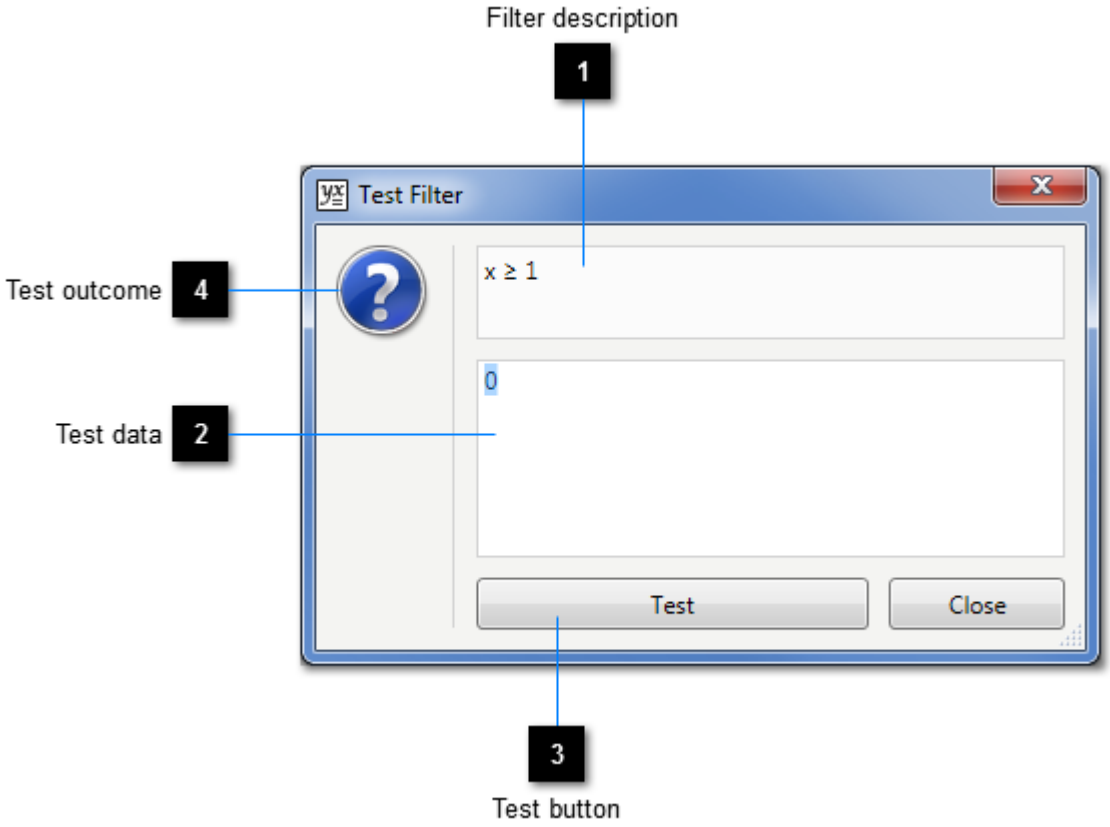
7 **Update Client**

 Sets the filter of the Filter Builder's current client component to be a copy of the Filter Builder's current composite filter.

8 **Add To Cache**

 Adds a copy of the Filter Builder's selected composite filter sub-node (and all its children) to the Composite Filter Cache.

Test Filter



1 Filter description

A description of the filter element or group being tested.

2 Test data

Enter the data to be tested here.

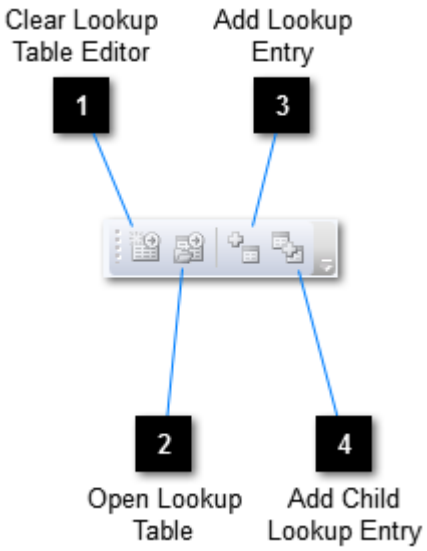
3 Test button

Click this button to perform the test. The visual test outcome indicator will be updated.

4 Test outcome

A visual representation of the result of the test.

Lookup Table toolbar



The Lookup Table toolbar works in conjunction with the Lookup Table Editor. You can show or hide this toolbar by selecting **View → Toolbars → Lookup Table** from the menu bar.

1 Clear Lookup Table Editor

Clears the Lookup Table Editor of its current lookup table.

2 Open Lookup Table

Opens a previously saved lookup table.

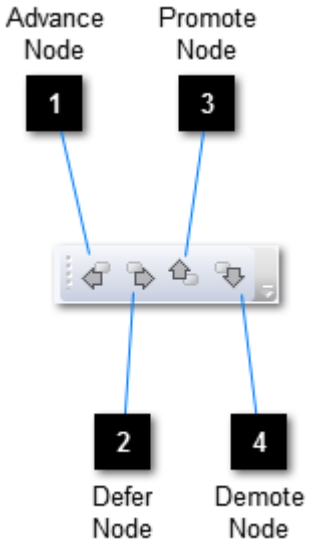
3 Add Lookup Entry

Adds a sibling lookup entry immediately after the currently selected lookup entry. The input parameter to which the entry refers is the Lookup Table Editor's currently selected input parameter.

4 Add Child Lookup Entry

Adds a child lookup entry to the currently selected lookup entry. The input parameter to which the entry refers is the Lookup Table Editor's currently selected input parameter.

Node Hierarchy toolbar



You can show or hide this toolbar by selecting **View → Toolbars → Node Hierarchy** from the menu bar.

1 Advance Node

Brings a node forward with respect to its siblings.

2 Defer Node

Sends a node backward with respect to its siblings.

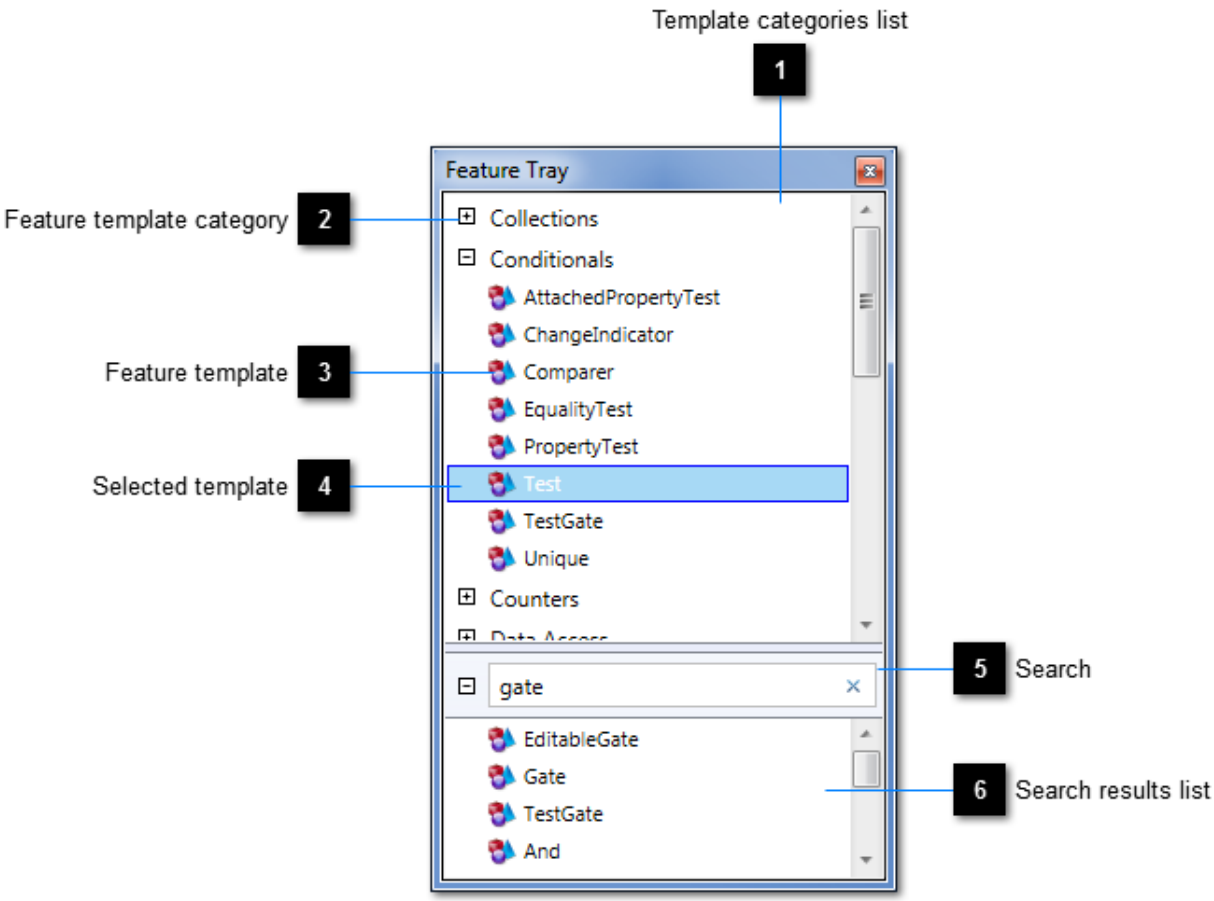
3 Promote Node

Moves a node such that it becomes a sibling of its current parent node.

4 Demote Node

Moves a node such that it becomes a child of a current sibling node.

Feature Tray



The Feature Tray contains all feature templates loaded into the Editor at start-up. These templates include all out-of-the-box templates installed with the Absyntax framework plus any detected third-party add-ins.

Select a template, then either drag it onto the design surface or, with the Feature Tray Tool selected, click on the design surface at the location where a new feature is to be added.

The Feature Tray is shown by selecting **View → Feature Tray** from the menu bar.

1 Template categories list

The list of all known feature template categories is presented here.

2 Feature template category

Templates are categorised for ease of access. A template may appear in any number of categories.

3 Feature template

A feature template contains all the information necessary to create a feature on the design surface. Hover the mouse over a template to show a tool-tip containing its description.

4 Selected template

The selected feature template is used by the Feature Tray Tool when creating new features on the design surface.

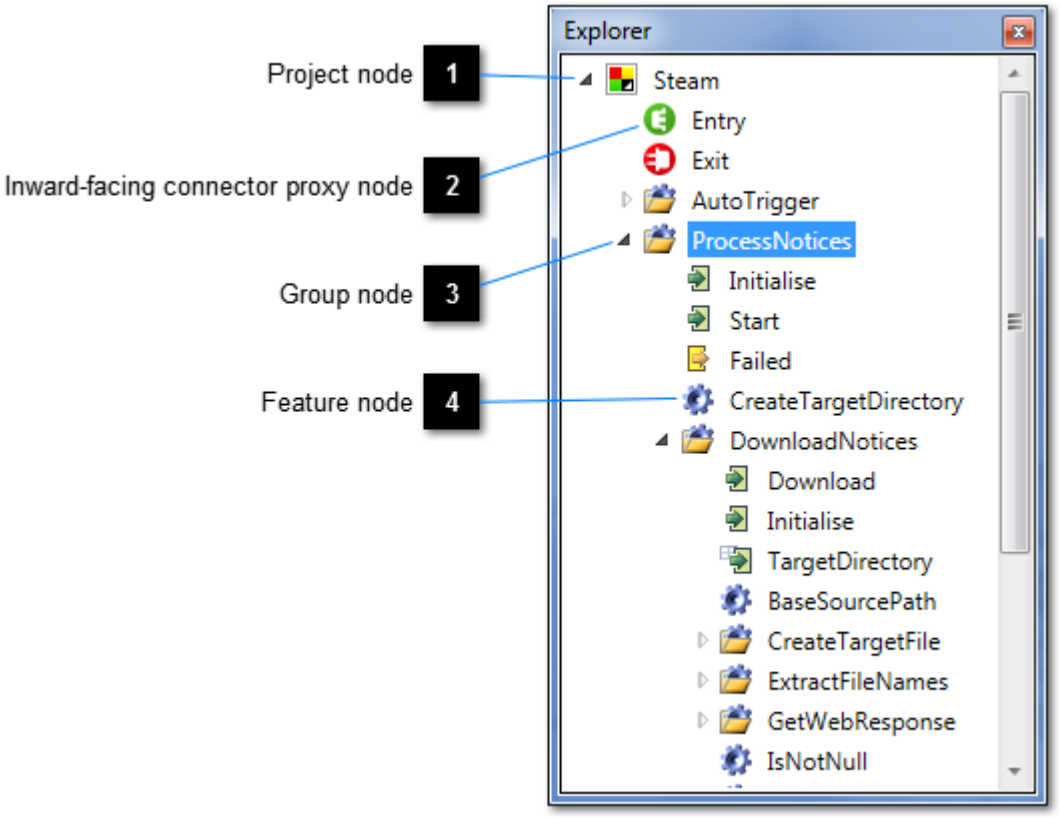
5 Search

Find feature templates using the search facility at the bottom of the Feature Tray. Words entered here are matched with a template's category, name, description and any culture-specific keywords associated with the template.

6 Search results list

This list contains the feature templates that are deemed to match the search words.

Explorer



The Explorer illustrates the hierarchical structure of the current project, containing nodes that represent distinct features and connector proxies. The selected component can be renamed by clicking on it. Alternatively, right-click on any node to show its context menu, from which other operations are available. Double-clicking a node will open the containing view and cause its viewport to pan and zoom to the component represented by the node.

The Explorer is shown by selecting **View → Explorer** from the menu bar or by clicking the Explorer button in the Standard toolbar.

1 Project node

Represents either the current project or a project nested within the current project.

2 Inward-facing connector proxy node

This is usually a representation of one of a group's connectors, but facing in towards the group's contained features so as to offer a means of connecting these features to the outside world.

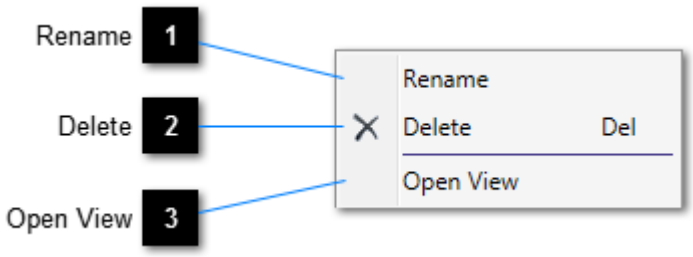
3 **Group node**

Represents a group of features.

4 **Feature node**

Represents a feature.

Explorer node context menu



1 Rename

Puts the selected node into edit mode, where it can be renamed.

2 Delete

Deletes the represented component from the project. If this option is disabled, the represented component cannot be deleted.

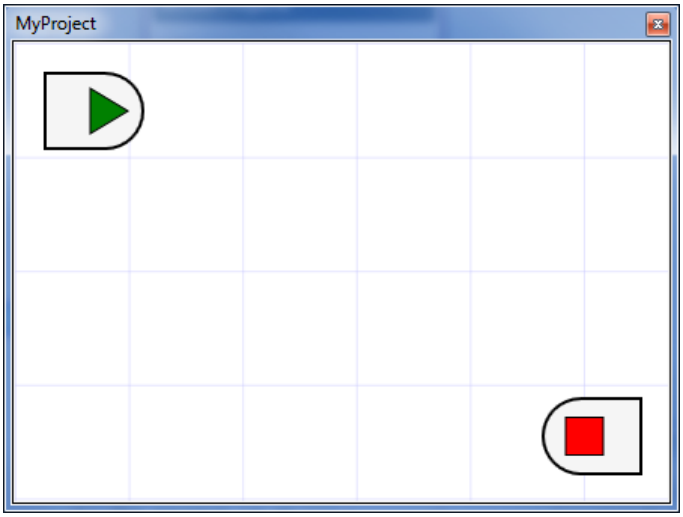
3 Open View

Opens the view corresponding to the feature group that this node represents. (If the node does not represent a feature group then this option is not displayed.)

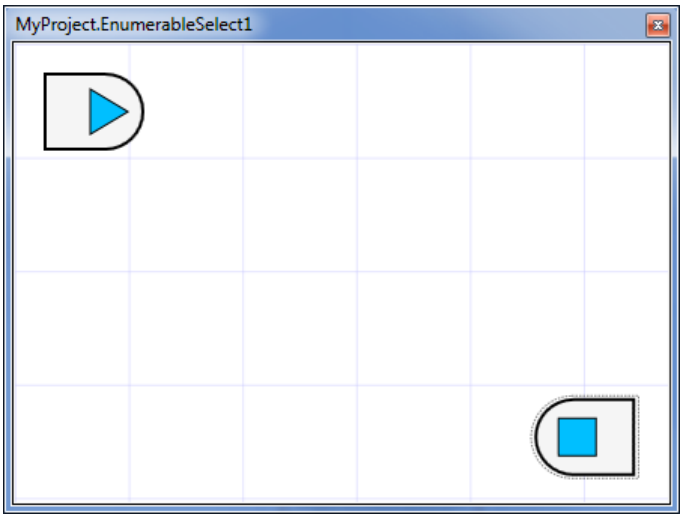
Feature Group Views

The current project and each feature group therein has a design surface offering you the ability to manage the content of the associated group. Design surfaces are presented in "views", with each view being a tabbed document in the Editor's main window.

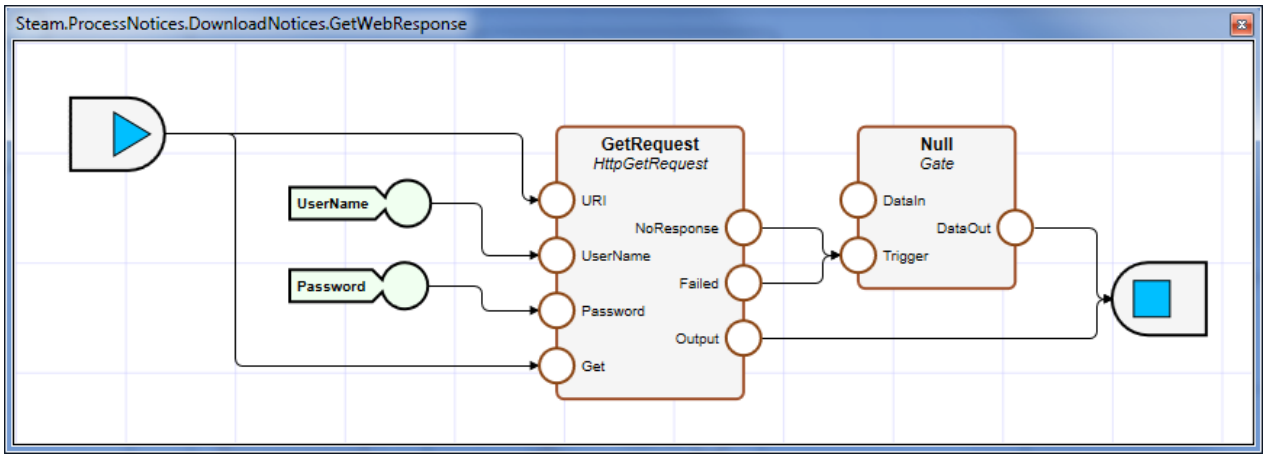
The view of an empty project appears as follows. The design surface includes connector proxies for the project's entry- and exit-points. These proxies are similar to those used by configurable feature groups, the only material difference being that a project's proxies cannot be deleted.



Some feature groups – for example, those that contain a user-defined operation to be applied to each item in a collection – have similar constraints and a similar appearance, as shown below.



The groups above are examples of groups that do not work in conjunction with the Connector Tray. In other words, their connector profiles are fixed. Hybrid groups exist – an example being the **MultiChannelGroup** – that support both immutable inward-facing connectors and user-definable connectors. Below is an example of such a group, in which user-defined connectors are included to provide initialisation capabilities to the group.

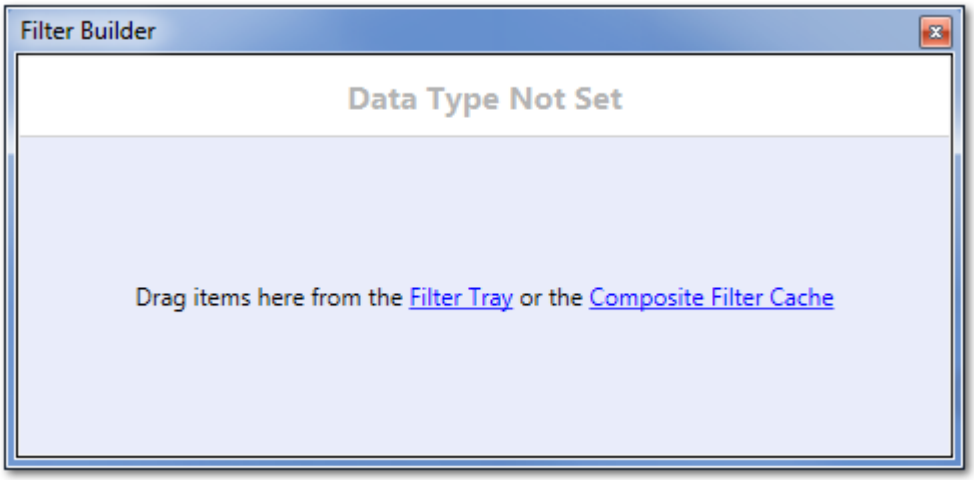


Filter Builder

The Filter Builder is a visual tool for creating, maintaining and testing Boolean expressions. The Filter Builder's visual embodiment of a Boolean expression is the "composite filter", which is an aggregation of one or more "filter elements". Each filter element encapsulates a simple Boolean test to be performed on some value. The aggregations are realised by "filter groups", each of which represents a Boolean operator such as "AND" and "OR" and can contain any number of elements and groups.

The Filter Builder is often used in conjunction with the Filter Tray and the Composite Filter Cache. It is shown by selecting **View → Filter Builder** from the menu bar.

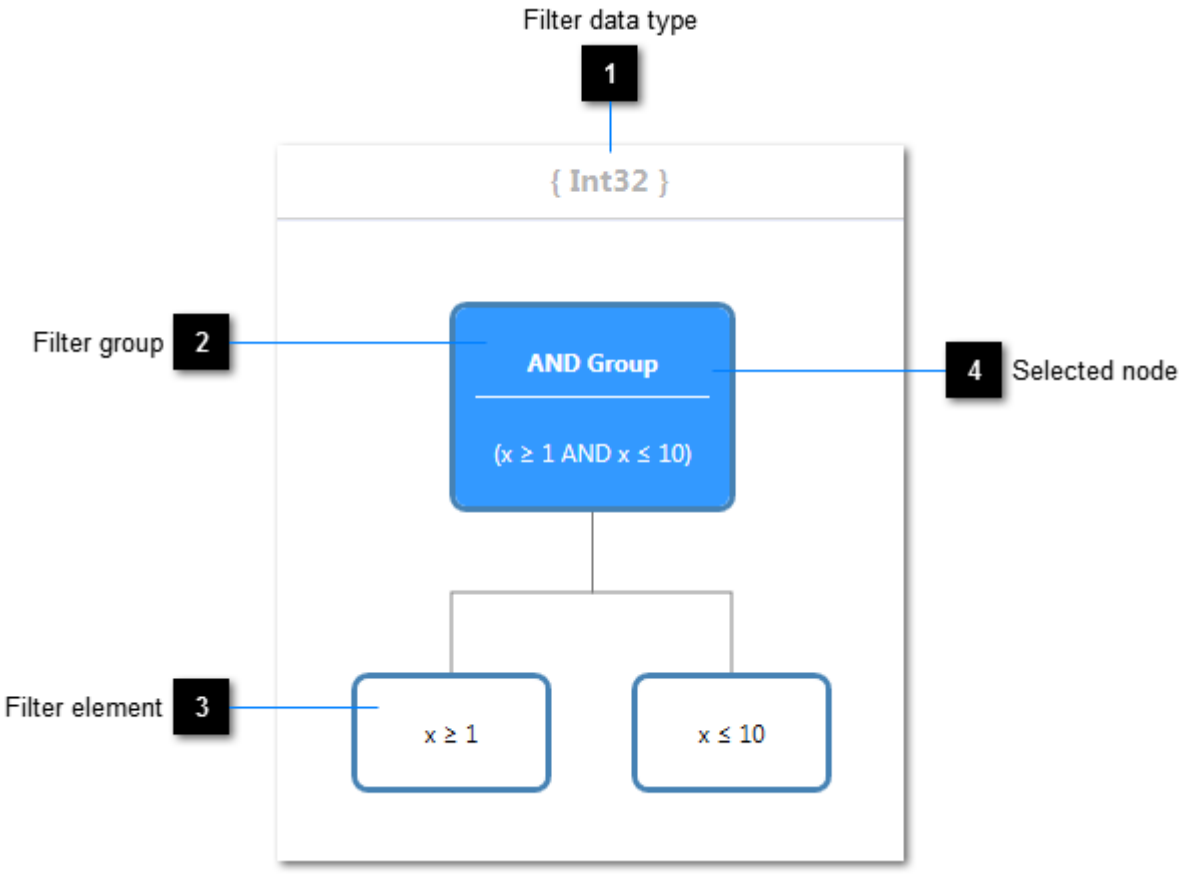
Creating a new filter



This is how the Filter Builder looks when it has no content and the data type has not been set. Filter elements and filter groups can be dragged onto the design surface from the Filter Tray; previously saved composite filters can be dragged onto the design surface from the Composite Filter Cache. Click on the links on the empty design surface to show their associated panels.

To create a new filter you must firstly specify the type of data that the filter will evaluate. With the Filter Builder as the active document, select **Filter → Filter Builder → Choose Data Type...** from the menu bar. This opens the Type Browser dialogue. Alternatively, drag a filter template from the Filter Tray onto the Filter Builder: this will also open the Type Browser if the chosen template can be applied to more than one data type.

Anatomy of a composite filter



All composite filters must have a root node. They should also have at least one filter element (which may be the root node itself), otherwise they would serve little purpose. They may also contain any number of filter groups.

When a composite filter is evaluated, the root node is tested against the supplied data. If the root node is a filter group, it returns a Boolean value based on the outcome of the tests of its child nodes. Any child feature group is also tested in this way.

In order to save a composite filter (or any part thereof), select the node that will be the root of the saved filter and drag it onto the Composite Filter Cache (or click **Add To Cache** from the Filter toolbar), where a copy of the sub-node hierarchy is stored. If, by the same stroke, you want to replace an existing cached filter, drop the dragged node onto it. You will be prompted to confirm such operations.

1 Filter data type

Denotes the type of data to be evaluated by the composite filter. This, in turn, determines the set of available filter templates and cached composite filters that may be used.

2 Filter group

A collection of filter elements and other filter groups. Filter groups allow you to create a hierarchical aggregate of individual tests. An evaluation of a filter group is really an evaluation of each of its children. The overall result depends on the nature of the filter group itself. For example, an AND group will return true only if *all* of its child nodes' constraints are satisfied by the test data. An OR group, on the other hand, will return true if *any* of its child nodes' constraints are satisfied.

In this example, the AND group will return true when evaluated with an integer of value in the range 1 to 10 inclusive.

The description of a filter group is a composite of the descriptions of each of its children.

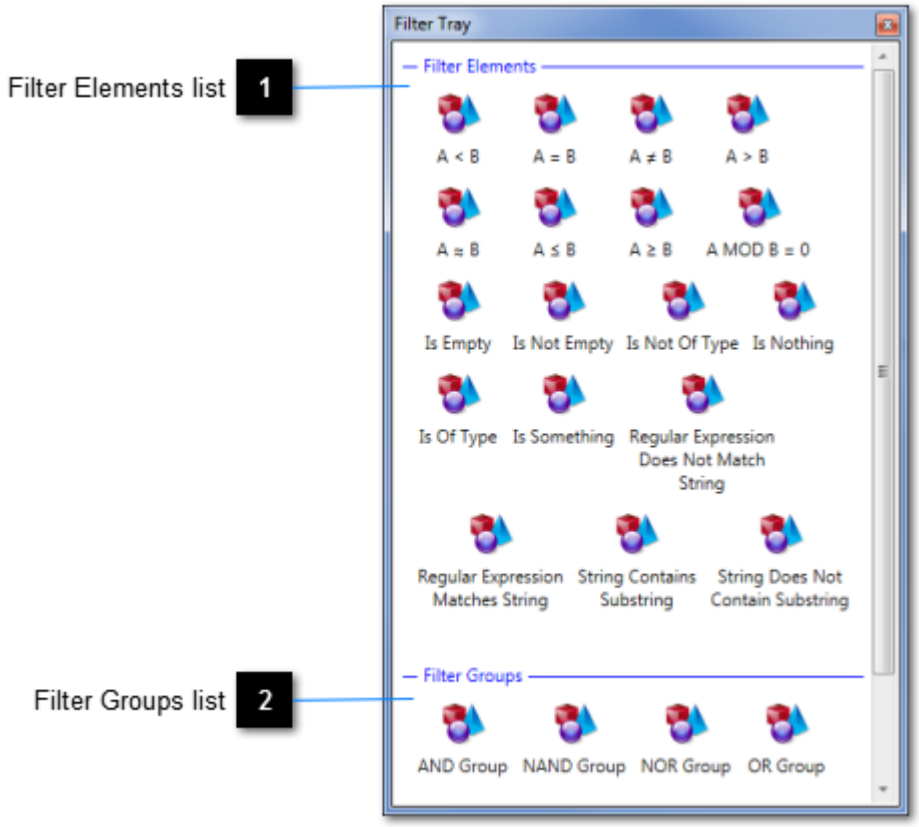
3 Filter element

A specific test on an item of data to be evaluated. To edit the operand value (if there is one), select the node. This will display the appropriate value editor for the current data type: use this to change the operand value. Alternatively, the Properties panel can be used: the relevant property is named TestOperand.

4 Selected node

The selected node is the target of Filter Builder operations (such as "test" and "move" actions).

Filter Tray



The Filter Tray contains all filter-related templates loaded into the Editor at start-up. These templates include all out-of-the-box templates installed with the Absyntax framework plus any detected third-party add-ins.

Select a template and drag it onto the Filter Builder's design surface in order to start editing a new composite filter. Or, drag a template onto an existing filter group in order to add a new element to that group.

The Filter Tray is shown by selecting **View → Filter Tray** from the menu bar or by clicking the hyperlink on the Filter Builder's empty design surface.

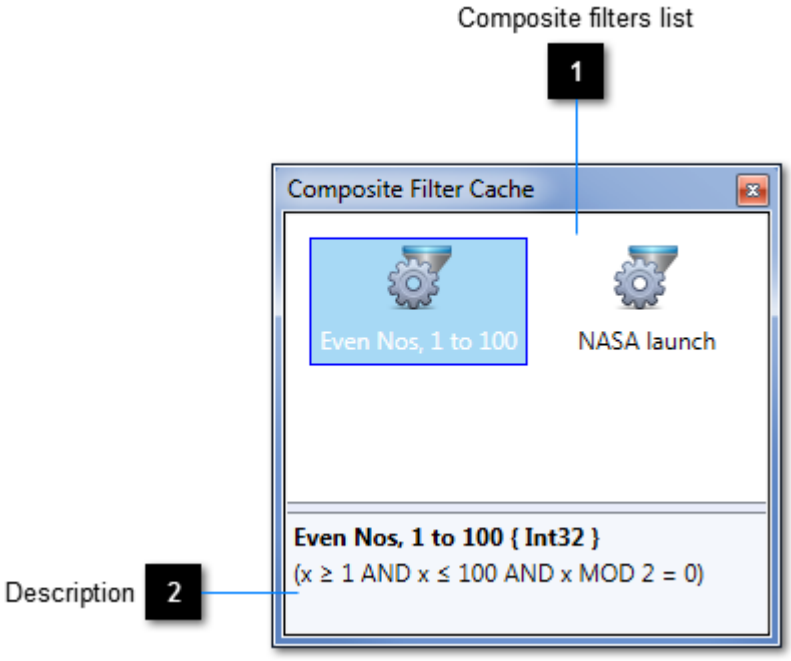
1 Filter Elements list

Lists all available filter element templates that match the Filter Builder's currently selected data type. If no data type has been specified then all available templates are listed.

2 Filter Groups list

Lists all available filter group templates.

Composite Filter Cache



The Composite Filter Cache contains all your saved composite filters.

Select a filter and drag it onto the Filter Builder's design surface in order to start editing a copy of it. Or, drag a filter onto an existing filter group in order to add a copy of it to that group. You can even drag a composite filter onto a feature that accepts filters (such as the **Test** feature), as long as the data type of the filter is the same as the data type predicated by the feature.

The selected filter can be renamed by clicking on it. Alternatively, right-click on any filter to show its context menu, from which other operations are available.

The Composite Filter Cache is shown by selecting **View → Composite Filter Cache** from the menu bar or by clicking the hyperlink on the Filter Builder's empty design surface.

1 Composite filters list

Lists all available cached composite filters that match the Filter Builder's currently selected data type. If no data type has been specified then all available filters are listed.

2 Description

A description of the selected composite filter.

Lookup Table Editor

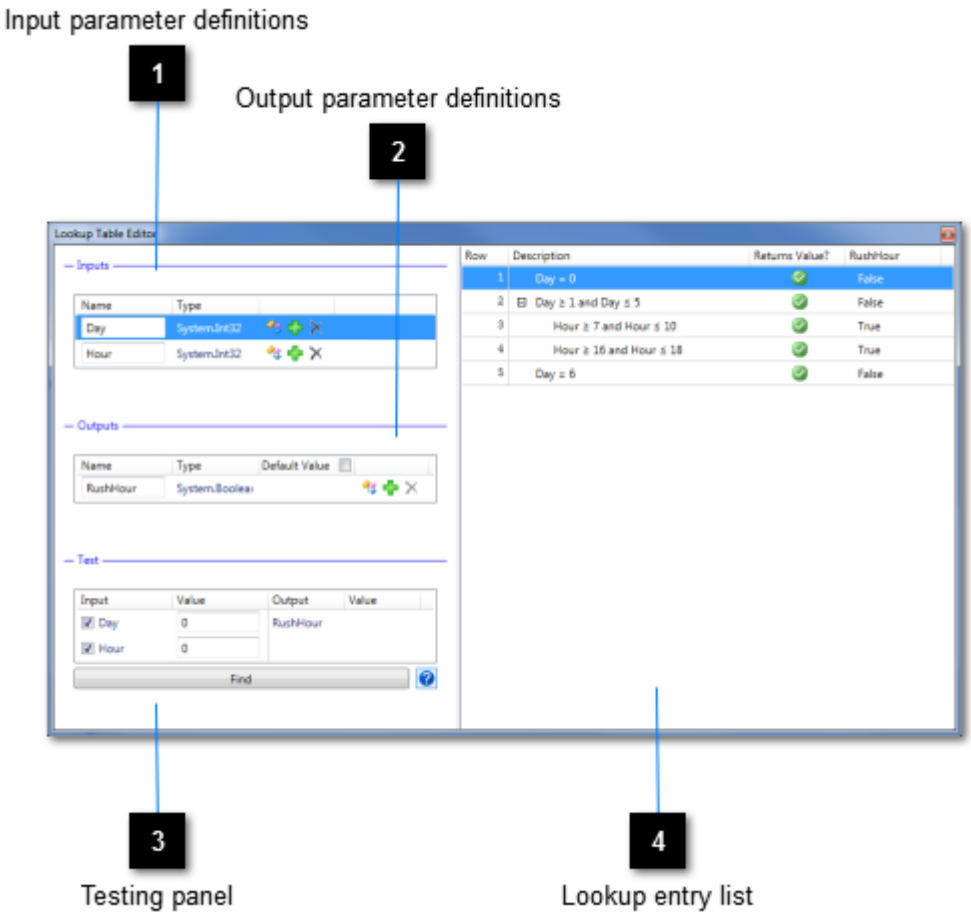
The purpose of a lookup table is to return a set of one or more output parameter values in response to the receipt of a set of one or more input parameter values. It is possible to achieve this kind of behaviour by other means (a database table, for example) but typically only if the set of input parameters is small and the domain of each parameter is likewise small. Consider using lookup tables if any of the following apply:

- you have multiple input parameter values to evaluate;
- the domain of any one input parameter value is large;
- you would like to use range-based tests or tests other than "equals";
- there are many combinations of input parameter values that return the same set of output parameter values;
- you need to test, interactively, that the expected set of output parameter values is being returned for a given set of input parameter values;
- you wish to reduce the likelihood of errors in your data definitions;
- your lookup data is likely to be modified, even occasionally.

Lookup tables are ideal for facilitating dictionary-style lookups (in which you supply a key datum and receive a value in return). They can be used to perform binary searches on a set of sorted data. They also support complex lookup scenarios involving multiple input parameter values and hierarchical tests. They are particularly advantageous in scenarios where it is acceptable for one or more input parameter values to be omitted. Once a lookup table has been created, it may be harnessed by a **Lookup** feature for use in a project.

The Lookup Table Editor is shown by selecting **View** → **Lookup Table Editor** from the menu bar.

Anatomy of a lookup table



1 Input parameter definitions

Use this section to define your lookup table's input parameters.

2 Output parameter definitions

Use this section to define your lookup table's output parameters.

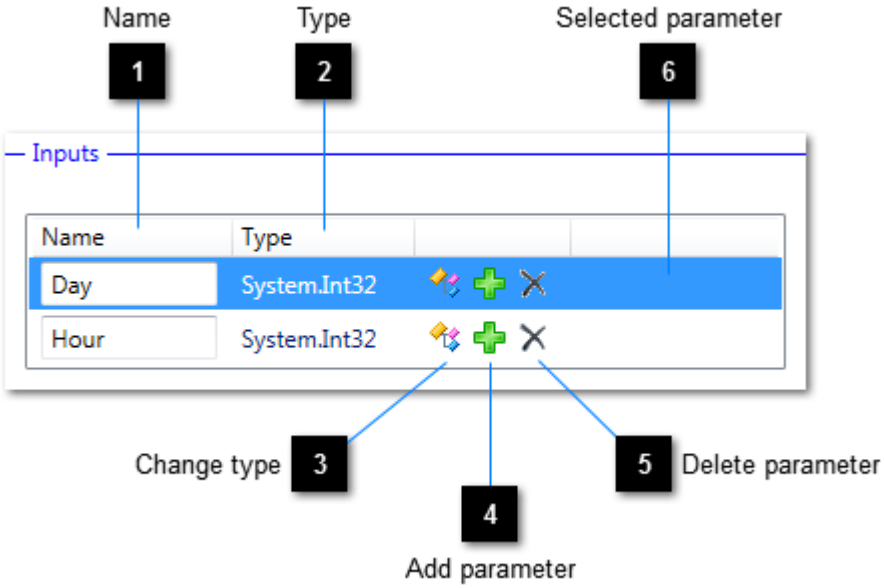
3 Testing panel

Use this section to test your lookup table.

4 Lookup entry list

Your lookup table's entries are defined here.

Input parameters



Use this panel to define the set of input parameters available to your lookup table.

1 Name

Use this field to define the name of each input parameter.

2 Type

Lists the data type of each input parameter.

3 Change type

Opens the Type Browser, allowing you to change the data type of the associated input parameter. The values of all existing lookup entries that refer to this input parameter will be converted.

4 Add parameter

Adds a new, default input parameter to the set.

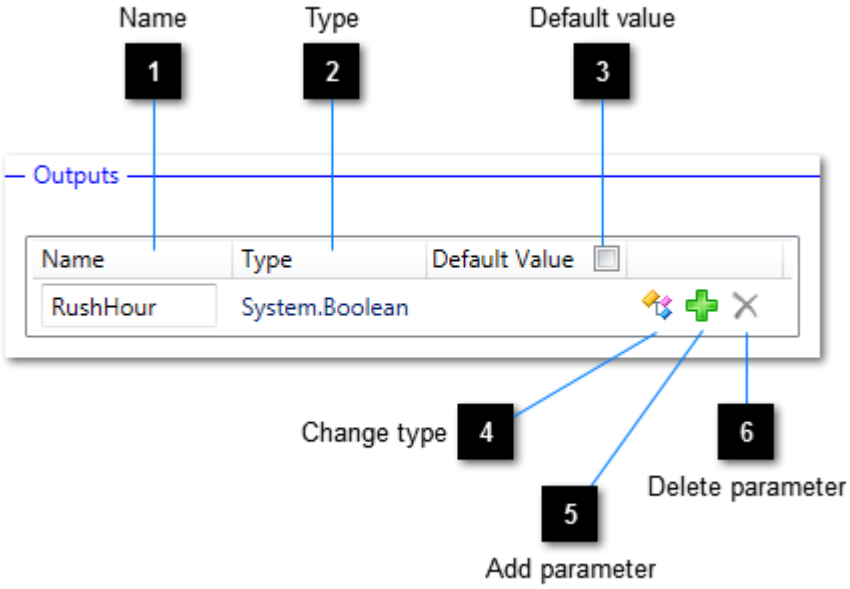
5 Delete parameter

Deletes the associated input parameter. All existing lookup entries that refer to this input parameter will be deleted. You will be prompted to confirm this action. Note that a lookup table must have at least one input parameter, so you will not be able to delete the sole remaining parameter.

6 **Selected parameter**

The selected parameter is used when adding a lookup entry.

Output parameters



Use this panel to define the set of output parameters for which your lookup table will return values.

1 Name

Use this field to define the name of each output parameter.

2 Type

Lists the data type of each output parameter.

3 Default value

Check this box if you want your lookup table to output a set of values in instances when none of the table's entries matches a particular set of input values. You will be able to specify the default value of each output parameter.

4 Change type

Opens the Type Browser, allowing you to change the data type of the associated output parameter. The values of all existing lookup entry output values that refer to this output parameter will be converted.

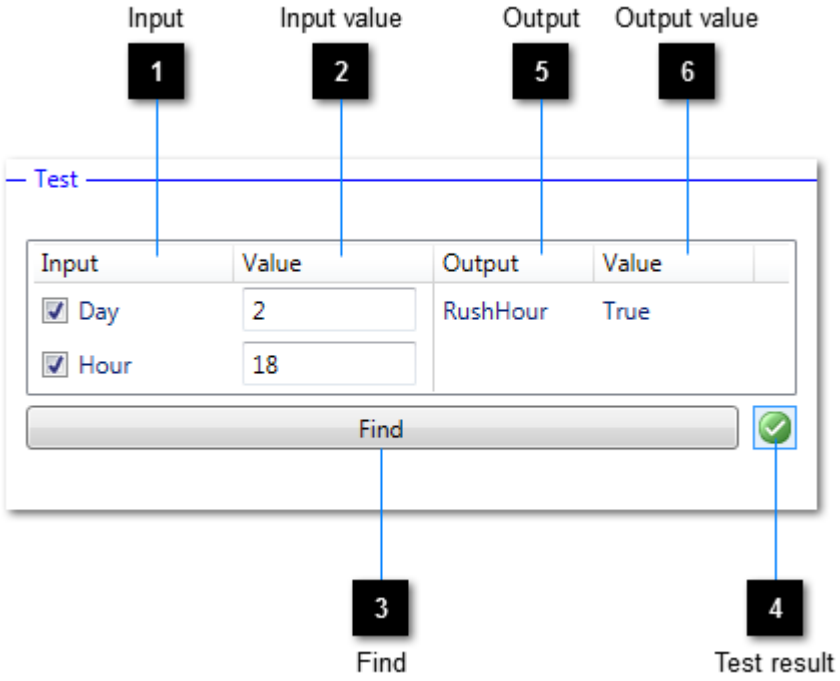
5 Add parameter

Adds a new, default output parameter to the set and creates a default value for each lookup entry defined as returning a value.

6 **Delete parameter**

Deletes the associated output parameter. All existing lookup entries will have values pertaining to this output parameter removed. You will be prompted to confirm this action. Note that a lookup table must have at least one output parameter, so you will not be able to delete the sole remaining parameter.

Testing



Use this panel to to test your lookup table. None of the edits you make in this panel affects the lookup table itself.

1 Input

When inputting values to a lookup table, you do not necessarily need to supply values for all input parameters. Uncheck those parameters for which values are not to be supplied in tests.

2 Input value

Use this field to specify the values of input parameters in tests.

3 Find

Click this button to pass the set of input parameter test values to the lookup table and obtain the result.

4 Test result

This button's graphic indicates the result of a test. A green tick indicates that either a lookup entry was matched with the input parameter test values or the lookup table's default values were returned (if such values are defined). A red "no entry" symbol indicates that no match was found and the lookup table did not return any default values.

This is a bistate button: when it is in the checked state (as shown here), a matching test will set the selected lookup entry to be the matching entry. You may want to uncheck the button if, for example, you are editing a large lookup table and do not want the currently selected lookup entry to be changed during a test.

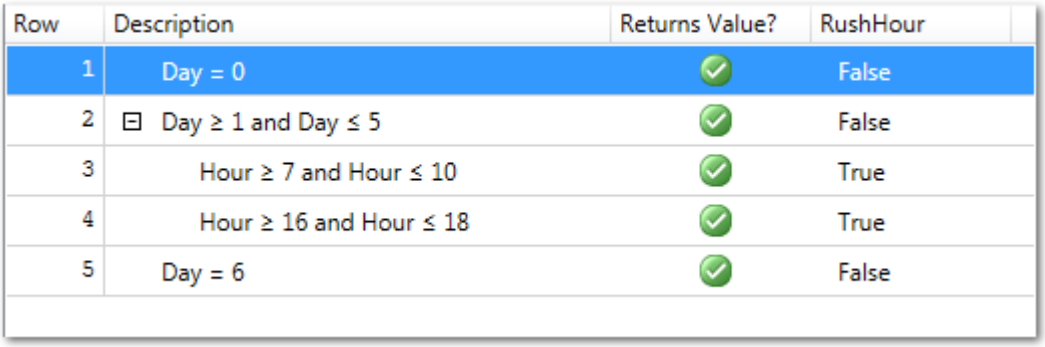
5 Output

Lists the lookup table's output parameters.

6 Output value

Lists the output parameter values obtained from the most recent test.

Lookup entries



A lookup table's capabilities are driven by its lookup entries. Each and every entry defines a comparison test to be performed on a specific input parameter value. For example, suppose a lookup table defines an integer input parameter named "Hour" and contains an entry that refers to said input parameter. If the entry's comparison test is "greater than or equal to 7", this means that the entry is deemed to be matched whenever it is presented with a set of input parameter values that include one named "Hour" with a corresponding integer value of 7 or higher.

A lookup entry may itself contain lookup entries. It may also define a set of output parameter values. If it does, the lookup table may return these values if the entry is matched with an input parameter value. Whether it *will* return them depends on whether the entry contains a child entry that also matches an input parameter value. In all instances, though, an entry can only be a match-candidate if both it and all of its ancestors satisfy their comparison tests.

Use the Lookup Table toolbar to add lookup entries to the list. A new entry is also added automatically (and placed in edit mode) when you press Return or Enter while editing the last lookup entry in the list. (This behaviour makes it easy to work solely with the keyboard.)

To edit a lookup entry, select it and click it to put it into edit mode. Alternatively, press Return or Enter and the selected entry will enter edit mode. To commit changes, press Return or Enter again. To cancel, press Esc. The Tab key cycles keyboard focus over the editable fields and, in conjunction with the Shift key, works in reverse too. Any lookup entry can be relocated by using the Node Hierarchy toolbar buttons or the relevant context menu items. Entries may also be cut, copied, pasted and deleted.

When a lookup operation is performed on a lookup table, each entry is tested in turn. Where an entry matches the incoming data, the entry's children are then tested (if there are any). When an entry is matched that defines a set of output parameter values, only its child entries are subsequently tested: if none of these child entries can be matched then the aforementioned output parameter values are returned by the lookup table and no further entry-testing takes place.

Example 1

In the above screenshot the lookup entries belong to a lookup table that serves to determine whether a specific hour of a specific day constitutes a rush-hour period. The entries are such that

if an input parameter value for "Day" is 0 then the lookup table will return a RushHour value of "False". This is because:

1. the row-1 lookup entry's comparison test is passed;
2. said lookup entry defines an output parameter value, this being "False";
3. said lookup entry has no children to test.

Example 2

Now suppose that an input parameter set is presented to the lookup table that contains a value of 3 for "Day". The lookup table will return a RushHour value of "False". This is because:

1. the row-1 lookup entry's comparison test fails, so the next sibling is tested;
2. the row-2 lookup entry's comparison test is passed;
3. said lookup entry defines an output parameter value, this being "False";
4. said lookup entry has two children to test but the absence of an "Hour" value in the input parameter set means that both tests fail.

Example 3

This time an input parameter set is presented that contains a value of 4 for "Day" and a value of 17 for "Hour". The lookup table will return a RushHour value of "True". This is because:

1. the row-2 lookup entry's comparison test is passed;
2. the comparison test of the second child of said lookup entry (row 4) is passed;
3. said lookup entry defines an output parameter value, this being "True";
4. said lookup entry has no children to test.

Example 4

This time an input parameter set is presented that contains a value of 6 for "Day" and a value of 10 for "Hour". The lookup table will return a RushHour value of "False". This is because:

1. the row-1 and row-2 lookup entries' comparison tests fail, so the next sibling (row 5) is tested;
2. rows 3 and 4 are ignored because they are children of an entry (row 2) for which the comparison test failed;
3. the row-5 lookup entry's comparison test is passed;
4. said lookup entry defines an output parameter value, this being "False";
5. said lookup entry has no children to test;
6. in this case the "Hour" input value proves redundant.

Example 5

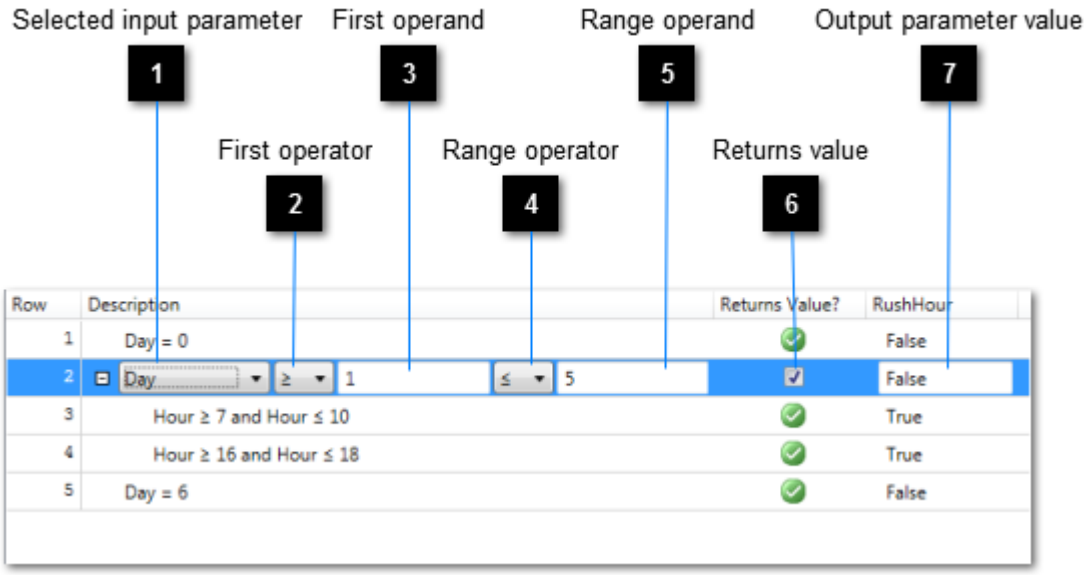
Finally, an input parameter set is presented that contains a value of 39 for "Day". The lookup table will return no match. This is because:

1. none of the rows' comparison tests is passed;
2. the lookup table itself does not define a default value.

A refinement

The lookup table used in these examples can be refined, as shown here.

Editing a lookup entry



The above example shows a lookup entry in the process of being edited.

1 Selected input parameter

This drop-down contains a list of the lookup table's defined input parameter values. Select the one that applies to the lookup entry.

2 First operator

This drop-down contains a list of all valid comparison operators. Select the one that reflects the nature of the test to be performed by the lookup entry.

3 First operand

This defines the value to be used by the first operator when the lookup entry is required to test the value of the selected input parameter.

4 Range operator

This drop-down contains a list of all valid "range" comparison operators. Select the one that reflects the nature of the range test to be performed by the lookup entry. If a range test is not required then select the blank item. This has the effect of clearing and disabling the range operand.

5 Range operand

This defines the value to be used by the range operator when the lookup entry is required to range-test the value of the selected input parameter.

6 Returns value

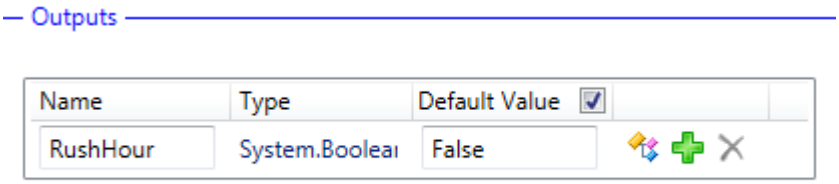
Check this box if output parameter values are to be defined for the lookup entry.

7 Output parameter value

This defines the value to be returned for the associated output parameter ("RushHour", in this example) in the event that the lookup entry is matched by the lookup table. A column is included for each output parameter defined by the lookup table.

Refined example

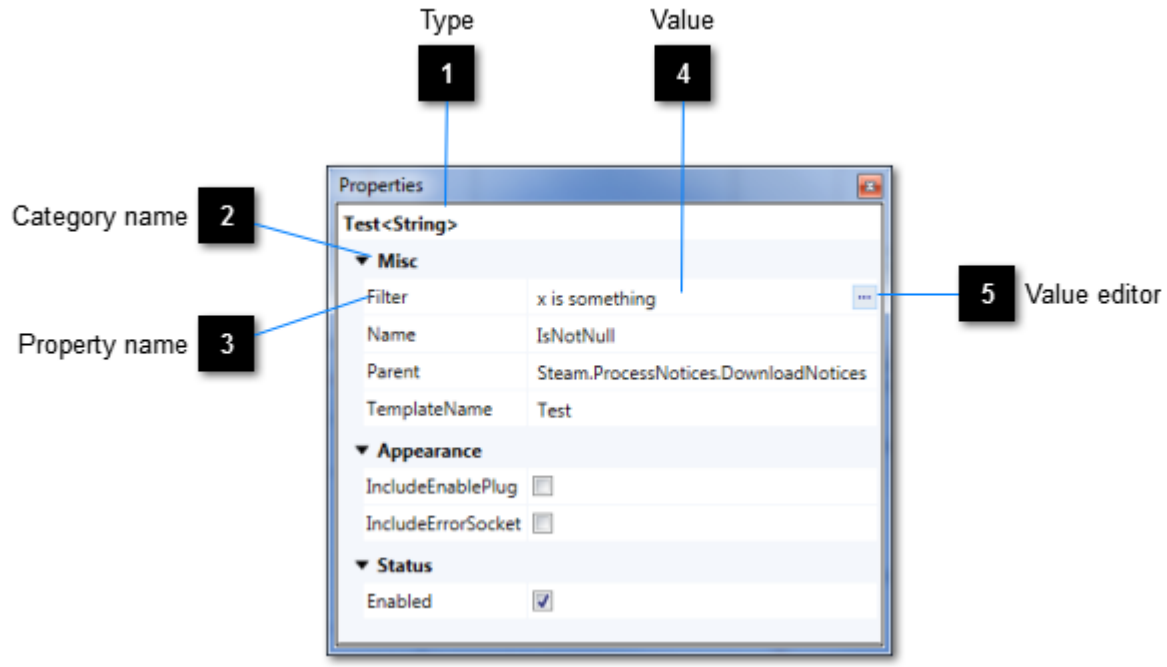
The lookup table used in the previous examples serves to illustrate the essential behaviour of all lookup tables. However, it is worth noting that this particular sample can be improved upon. Similar results can be achieved by deleting two of the five entries, configuring another not to return a value and defining a default value for the table, as shown below. Of course, with this approach the lookup table is guaranteed to return a match every time (meaning that example 5 would instead return a value). Be careful when declaring a lookup table as having a default value. In example 5, an input value of 39 is invalid (the domain of day values is 0 through 6) and in this case it may be more appropriate to return no match.



Row	Description	Returns Value?	RushHour
1	⊟ Day ≥ 1 and Day ≤ 5	⛔	
2	Hour ≥ 7 and Hour ≤ 10	✔	True
3	Hour ≥ 16 and Hour ≤ 18	✔	True

With the structure shown above, if either an incoming "Day" value is not in the range 1 to 5 or an incoming "Hour" value is not any of 7, 8, 9, 10, 16, 17 or 18 then the lookup table's default value ("False" in this case) is returned.

Properties



The Properties panel works on behalf of the currently selected object or objects in the active document or another of the Editor's panels. If a single object is selected, the panel will display the object's properties and their values. If multiple objects are selected, the panel will display those properties that are common to all of the objects. In such cases, common values will also be displayed.

Although the descriptions in this section relate to the Editor's main property grid, some of the Editor's dialogue windows also make use of property grids and they all operate in the same way.

The Properties panel is shown by selecting **View → Properties** from the menu bar or by clicking the Properties button in the Standard toolbar.

1 Type

Displays the type of the selected target object or objects. If multiple different types are selected, this label displays "Various".

2 Category name

Properties are categorised. This label displays the name of a category.

3 Property name

The name of one of the target object's properties. Hover the mouse over a property's name to show a tool-tip containing its description.

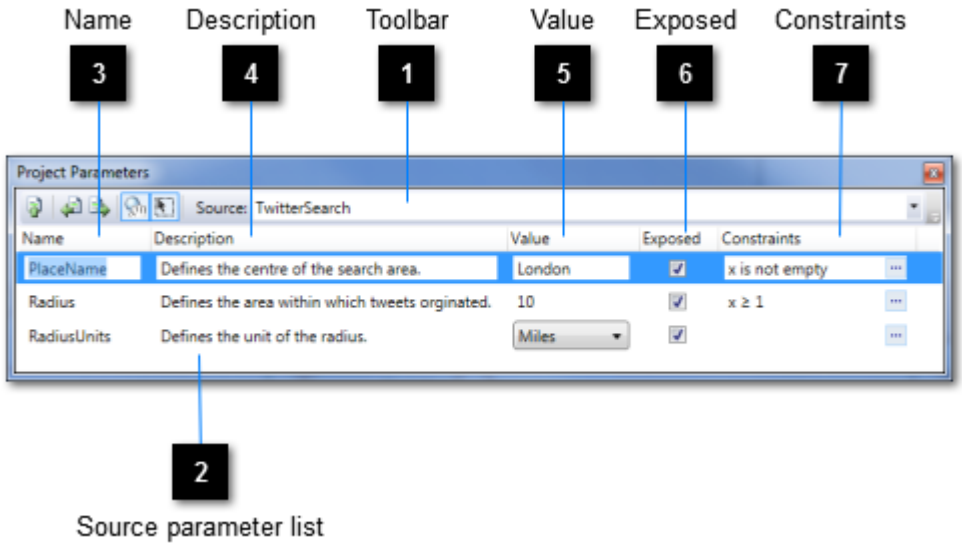
4 **Value**

A representation of the value of one of the target object's properties. If the property value is not read-only, you may change its value here. If multiple objects are selected, the property value for each selected object will be changed.

5 **Value editor**

Press this button to display a dialogue editor that will allow you to maintain the value of the associated property. This button is only visible for properties whose values cannot readily be expressed using a string value alone.

Project Parameters



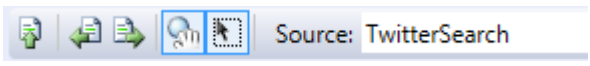
A project declares zero or more "parameters", these being preset values that the project's various features can use on demand. Such parameters are expressed by the **EditableGate** feature, and the inclusion of an EditableGate instance in a project is equivalent to the inclusion of a parameter.

A parameter is deemed to belong to its first parent project. Thus the root project and every project instance nested therein may each declare parameters. In this way a project is considered to be a "source" of parameters. The Project Parameters panel makes use of this when displaying collections of parameters.

Parameters have names and values, and the EditableGate feature supports this paradigm. The Project Parameters panel makes it easy to manage parameter attributes in one place. Furthermore, parameters may be harnessed by other features (namely the **ParameterInitialiser** and the **ParametersPrompt**) to perform parameter-driven operations at runtime.

The Project Parameters panel is shown by selecting **View → Project Parameters** from the menu bar or by clicking the Project Parameters button in the Standard toolbar.

1 Toolbar



The toolbar supports actions in respect of the parameter list.

2 Source parameter list

Displays the list of parameters found for the currently selected source.

3 **Name**

Use this field to assign a name to each parameter.

4 **Description**

Use this field to assign a description to each parameter. The description will help users to understand the purpose of the parameter.

5 **Value**

Use this field to assign a value to each parameter.

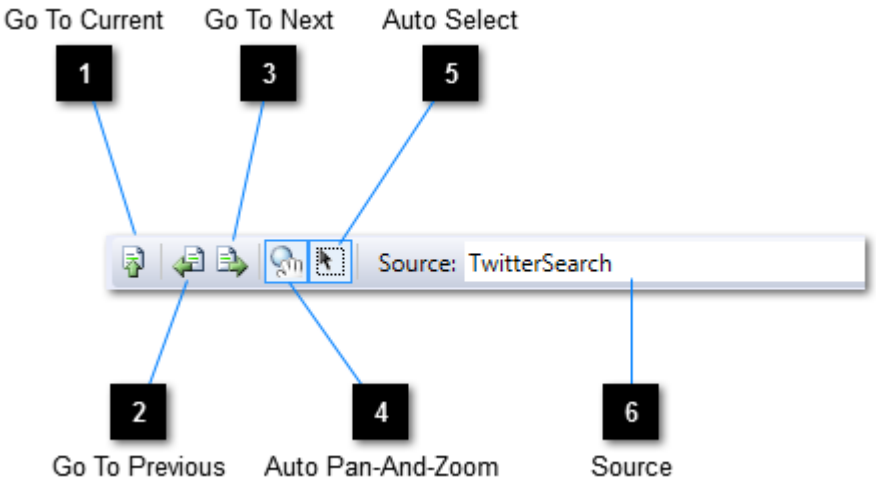
6 **Exposed**

Use this field to indicate whether a parameter is to be exposed at runtime via the **ParametersPrompt** feature.

7 **Constraints**

Use this field to apply constraints to the value of any parameter. At runtime, if the user is prompted to override parameter values via the **ParametersPrompt** feature, all constrained parameters' values must satisfy their constraints before the overridden values will be accepted.

Project Parameters toolbar



1 Go To Current

Actions the currently selected parameter in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

2 Go To Previous

Selects the previous parameter in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

3 Go To Next

Selects the next parameter in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

4 Auto Pan-And-Zoom

When selected, causes the actioned parameter to be brought into view and scaled such that it fills the viewport.

5 Auto Select

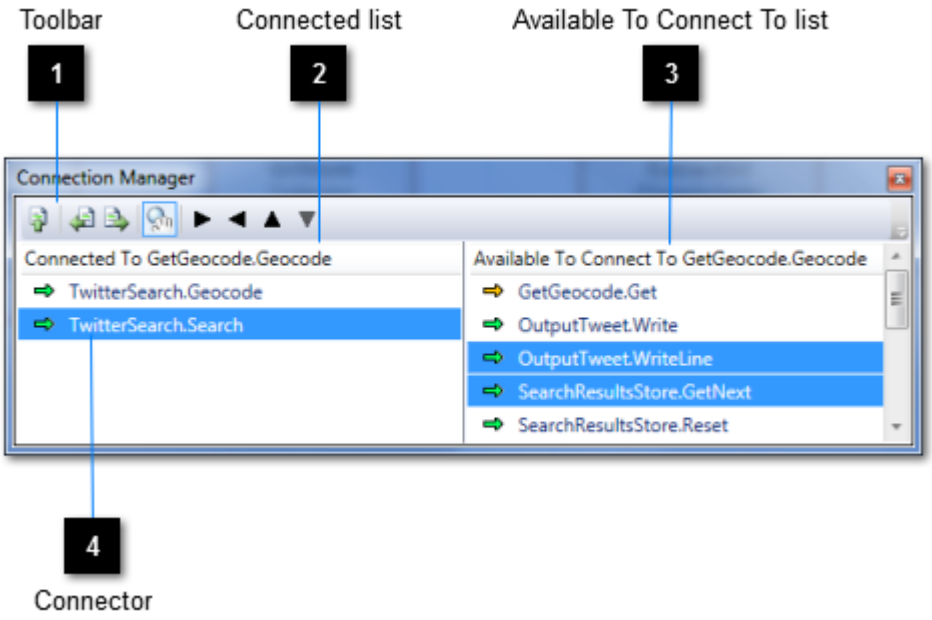
When selected, causes the actioned parameter to be selected.

6 Source

Source: TwitterSearch

From the drop-down, select the project that is to be the source of parameters for the parameter list.

Connection Manager



As well as being able to create connections using the Connection Tool, you can use the Connections Manager to achieve the same results. If you want to make multiple connections to the same connector or the design surface contains a lot of features, the Connections Manager makes the task much easier. It also helps to visualise the order in which a socket's connections are triggered, something that is fundamental in determining your project's overall behaviour.

From the design surface, select a single connector. The Connections Manager will then display a list of those connectors currently connected to the selected connector and a list of those connectors that may be connected to the selected connector.

The Connection Manager panel is shown by selecting **View → Connection Manager** from the menu bar or by clicking the Connection Manager button in the Standard toolbar.

1 Toolbar

The toolbar supports actions in respect of both lists.

2 Connected list

A list of all connectors currently connected to the selected connector.

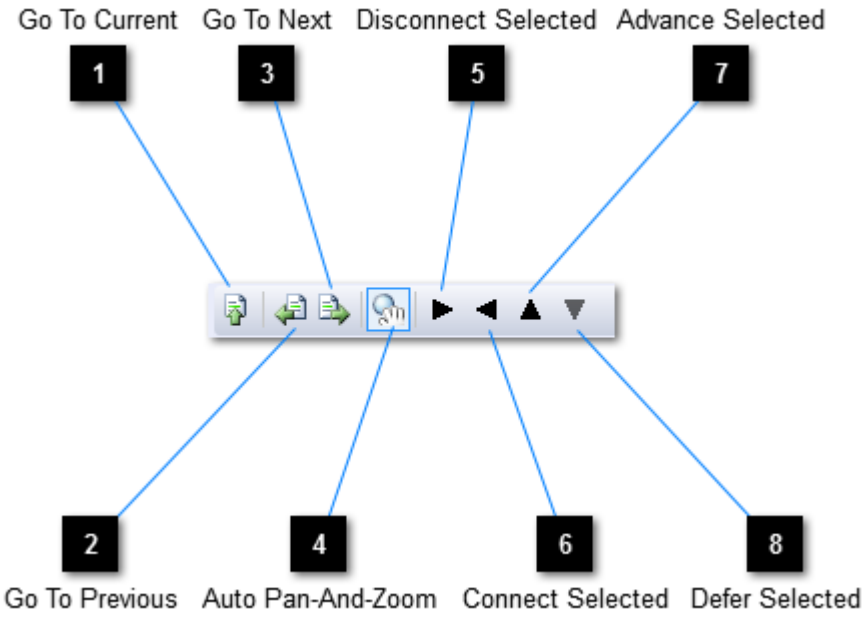
3 Available To Connect To list

A list of those connectors that may be connected to the selected connector.

4 Connector

Combines a connector's user-assigned name and its host feature's name. An arrow indicates the direction of communication (left to right = outgoing signals or data, right to left = incoming signals or data); the arrow's colour indicates whether transmission is guaranteed.

Connection Manager toolbar



1 **Go To Current**

Actions the currently selected item in the Connected list in accordance with the selections for Auto Pan-And-Zoom.

2 **Go To Previous**

Selects the previous item in the Connected list in the list and actions it in accordance with the selections for Auto Pan-And-Zoom.

3 **Go To Next**

Selects the next item in the Connected list in the list and actions it in accordance with the selections for Auto Pan-And-Zoom.

4 **Auto Pan-And-Zoom**

When selected, causes the actioned connection to be brought into view and scaled such that it fills the viewport.

5 **Disconnect Selected**

Breaks connections between the selected connector and those connectors selected in the Connected list.

6 **Connect Selected**

 Creates connections between the selected connector and those connectors selected in the Available To Connect To list.

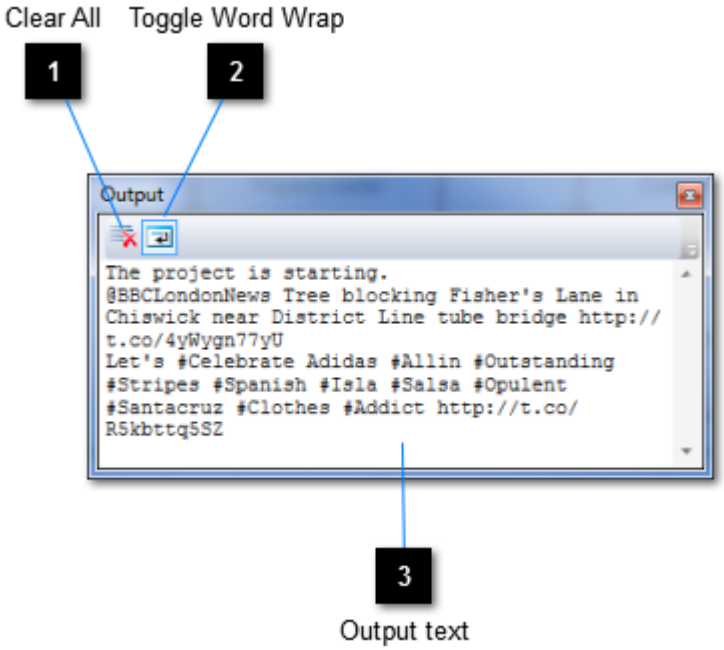
7 **Advance Selected**

 Promotes those connectors selected in the Connected list such that connections between them and the selected socket are triggered earlier in the socket's trigger sequence.

8 **Defer Selected**

 Demotes those connectors selected in the Connected list such that connections between them and the selected socket are triggered later in the socket's trigger sequence.

Output



The Output panel is used by the Editor to display text relating to a project's execution. Runtime information and error messages are displayed here, coupled with any text directed at the **Output** feature.

Any strings written to the console at runtime using either `Console.Write` or `Console.WriteLine` will be redirected to the output panel.

The Output panel is shown by selecting **View** → **Output** from the menu bar or by clicking the Output button in the Standard toolbar.

1 Clear All

Clears the output panel of text.

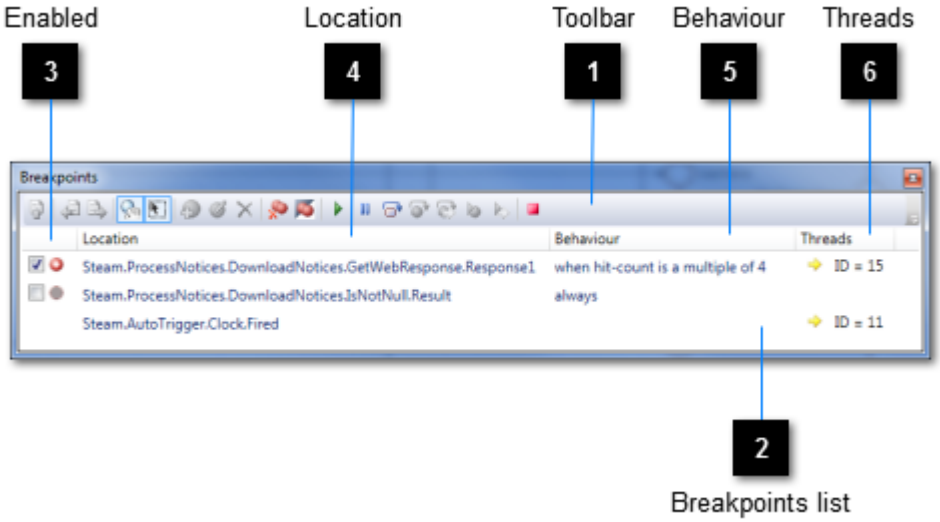
2 Toggle Word Wrap

When selected, causes text in the output panel to be wrapped.

3 Output text

Displays all output text from an executing project.

Breakpoints



It is often important to be able to see what your project is doing while it is doing it. This will enable you to make more informed changes to a project and more quickly realise the project's objectives.
To this end, the Editor allows you to pause a project while it is running. After a pause request is issued, each thread is halted when it next emerges from a feature. At this point, any data that the thread is carrying can be examined in the Runtime Data panel and, under certain circumstances, may even be changed.

Breakpoints can be set on sockets. A breakpoint serves to block threads in much the same way as a pause request but with the advantage that it is location-specific. When a thread is blocked by a breakpoint, all other threads will also be paused when they next emerge from their features.
To set a breakpoint you will need to open the view that contains the target socket. Then, with the Pointer Tool selected, double-click the target socket on the design surface. Alternatively, with the target socket or sockets selected, from the menu bar choose **Debug → Breakpoint → Set Breakpoint** or from the context menu of either the Pointer Tool or Connection Tool, choose **Debug → Breakpoint**. At runtime you may also set a breakpoint at a location where there is a paused thread, by clicking Set Breakpoint on the toolbar.

By default, breakpoints always block threads. However, they can be configured to block only when certain conditions arise. These conditions are functions of either:

- the number of times a thread has passed through a breakpoint;
- the state of the data being carried by a thread.

As well as enabling you to pause threads, the Editor supports thread-stepping and thread-resumption for:

- individual threads;

- all threads waiting at selected breakpoints;
- all project threads.

The Breakpoints panel is shown by selecting **View** → **Breakpoints** from the menu bar or by clicking the Breakpoints button in the Standard toolbar.

1 **Toolbar**

The toolbar supports actions in respect of the breakpoints list.

2 **Breakpoints list**

A list of all breakpoints. At runtime this may include locations without breakpoints at which threads are paused.

3 **Enabled**

A checked box indicates an enabled (i.e. active) breakpoint. An unchecked box indicates a breakpoint that will not block threads.

4 **Location**

A description of the location of the associated breakpoint.

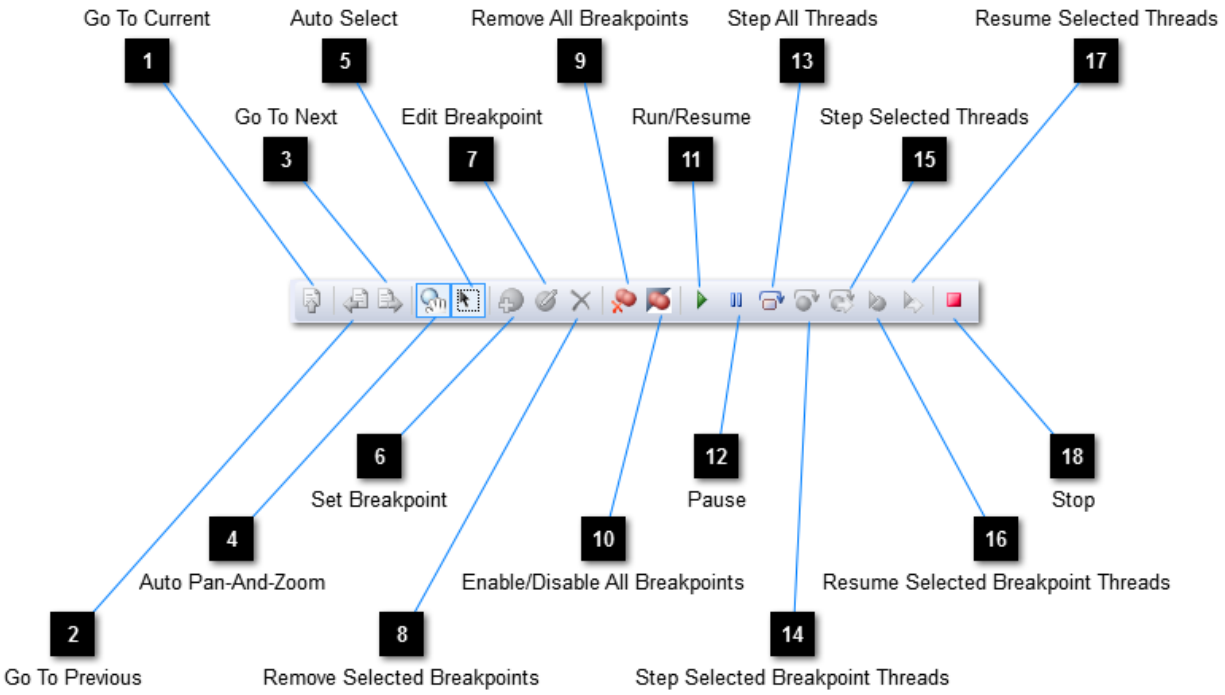
5 **Behaviour**

A description of the behaviour of the associated breakpoint.

6 **Threads**

A list of the identifiers of threads currently paused at the associated breakpoint.

Breakpoints toolbar



1 Go To Current

 Actions the currently selected breakpoint in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

2 Go To Previous

 Selects the previous breakpoint in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

3 Go To Next

 Selects the next breakpoint in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

4 Auto Pan-And-Zoom

 When selected, causes the actioned breakpoint to be brought into view and scaled such that it fills the viewport.

5 Auto Select

 When selected, causes the actioned breakpoint to be selected.

6 **Set Breakpoint**

 Sets a breakpoint at the location of the paused thread.

7 **Edit Breakpoint**

 Opens the Breakpoint Editor, allowing you to manage the conditions under which the selected breakpoint will interrupt threads.

8 **Remove Selected Breakpoints**

 Removes the selected breakpoints from their locations.

9 **Remove All Breakpoints**

 Removes all breakpoints from their locations.

10 **Enable/Disable All Breakpoints**

 Toggles the enabled state of all breakpoints.

11 **Run/Resume**

 Starts executing the current project or resumes all of a project's paused threads.

12 **Pause**

 Pauses all threads in the currently executing project.

13 **Step All Threads**

 Steps all paused threads in the currently executing project.

14 **Step Selected Breakpoint Threads**

 Steps all threads at the selected breakpoints.

15 **Step Selected Threads**

 Steps all selected threads.

16 **Resume Selected Breakpoint Threads**

 Resumes all threads at the selected breakpoints.

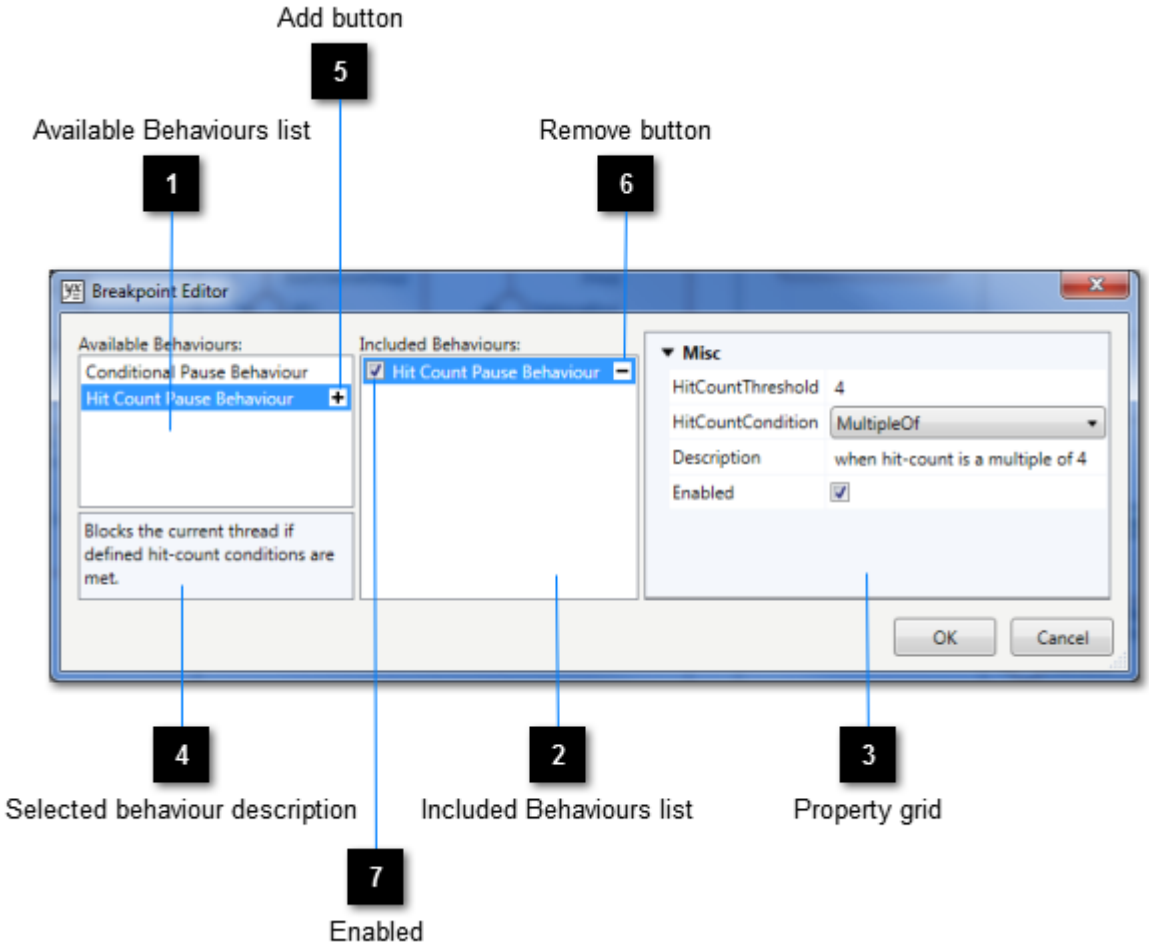
17 **Resume Selected Threads**

Resumes all selected threads.

18 **Stop**

Aborts the currently executing project.

Breakpoint Editor



The Breakpoint Editor dialogue is used manage the interrupt behaviours associated with a breakpoint. Such behaviours collectively determine whether a thread should be blocked: all active (i.e. enabled) behaviours must be satisfied in order for this to happen.

Interrupt behaviours may be added and removed. Included behaviours may also be configured using a property grid.

1 Available Behaviours list

A list of the behaviour templates available for selection.

2 Included Behaviours list

A list of the behaviours already included with the breakpoint being edited.

3 Property grid

The set of the selected included behaviour's properties and their values.

4 Selected behaviour description

A description of the selected behaviour template.

5 Add button

 Click this button to add an instance of the associated behaviour template to the breakpoint being edited. This has the effect of adding a new, default instance of the behaviour type to the Included Behaviours list.

When adding a new Conditional Pause Behaviour, the Type Browser will be opened so that you can specify the type of data that the behaviour is to evaluate. Note that there is currently no means of coupling the type of data to be evaluated with the data type of the associated socket: if you choose a data type that does not match that of the socket, the test encapsulated by the behaviour will not be performed.

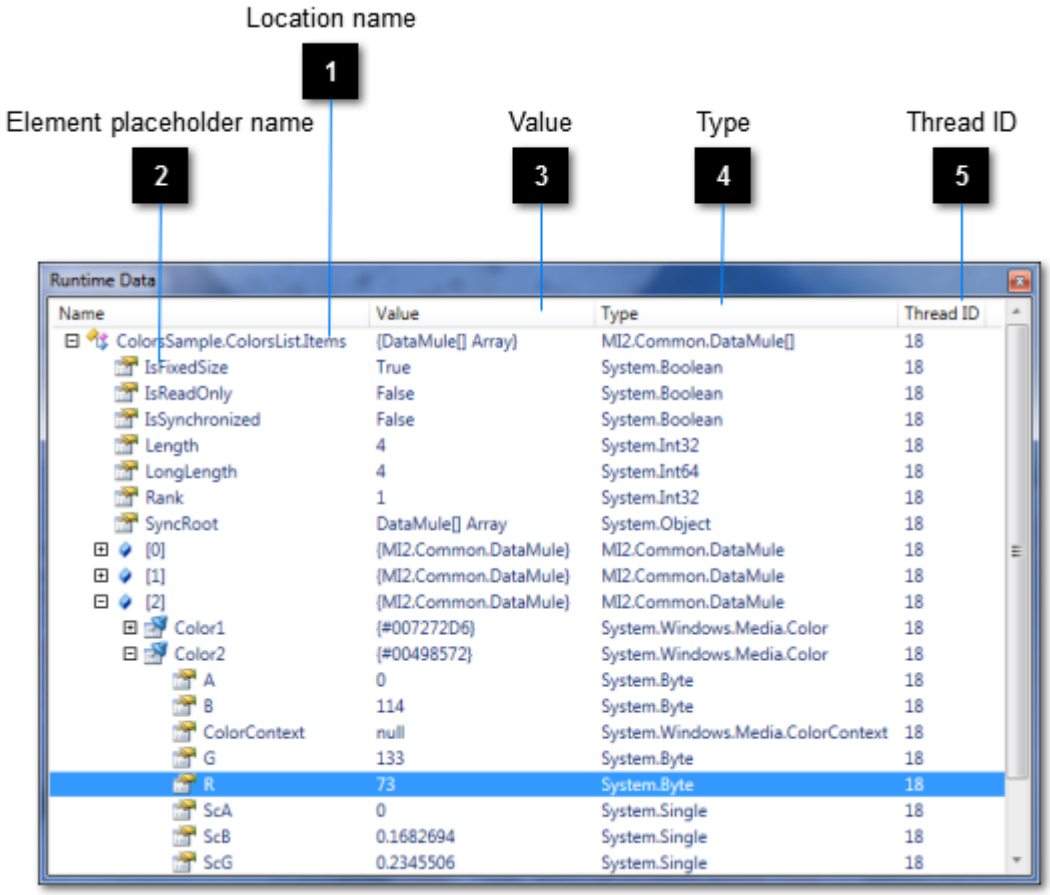
6 Remove button

 Click this button to remove the selected behaviour from the breakpoint being edited.

7 Enabled

When checked, the associated behaviour is active, meaning that it will be evaluated when the runtime is determining whether to block a thread. To have the behaviour ignored, uncheck this box.

Runtime Data



Where a data-carrying thread is paused during project execution, its data may be examined interactively by using the Runtime Data panel. For each such thread there will be a location name (i.e. the name of the socket that it is currently paused at).

As explained here, objects can be composed of other objects, so we can think of data as a hierarchy of information. The Runtime Data panel reflects the hierarchical nature of object composition by allowing you to "expand" data into its constituent parts, as shown in the above example.

The Runtime Data panel is shown by selecting **View → Runtime Data** from the menu bar or by clicking the Runtime Data button in the Standard toolbar.

1 Location name

The name of the location at which the root object is currently sited.

2 Element placeholder name

The name of an element of the parent object. The various icons used to portray these elements are:


Denotes a property of the parent object.

Denotes an enumerated item (i.e. an item that belongs to a collection). Such elements are accompanied by names of the form [*n*], where *n* is a unique, sequential, zero-based index number.


Denotes an attached property of the parent object.

3 Value

A textual representation of the associated element's value, or "null" if no element exists. If the element's placeholder is not read-only, the element may be changed by clicking on this field.

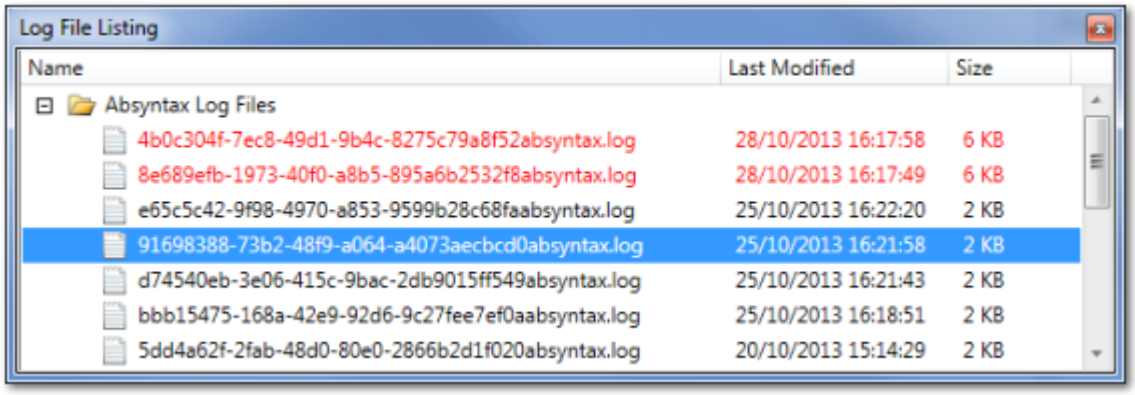
4 Type

The associated element placeholder's allowed data type.

5 Thread ID

The identifier of the thread with which the data is associated. All properties of the root object share the same thread as that of the root object itself.

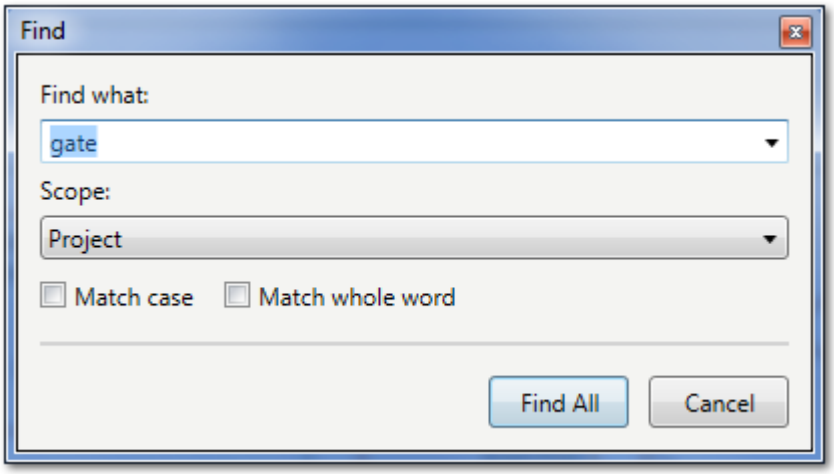
Log File Listing



Log files are produced when a project is executed. Entries may be written to these log files during execution and, if they are, you can easily view the contents of the files by opening the Log File Listing (select **View → Log File Listing** from the menu bar). The list of files is presented in order of descending last-modified date. Entries in red indicate files that have changed since the Editor was opened. Double-click on any entry to open its contents as a document.

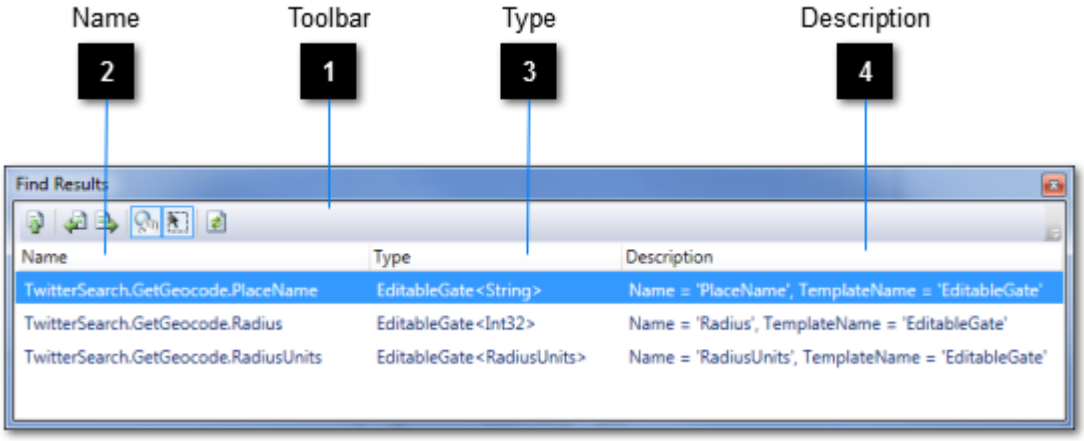
The files themselves are accessible via the file system. The path of their containing directory may be obtained by hovering the mouse over the folder item in the list: the resulting tool-tip yields the path.

Find



You can search for items in the current project that match your text-based search criteria. To open the Find dialogue, press **Ctrl-F** or select **Edit → Find...** from the menu bar. Clicking Find All performs the search and displays the results in the Find Results panel.

Find Results



The Find Results panel contains a list of items in the current project that match your text-based search criteria. It is shown by either performing a search or selecting **View → Find Results** from the menu bar.

1 Toolbar

The toolbar supports actions in respect of the list of found items.

2 Name

The name of the associated matched item.

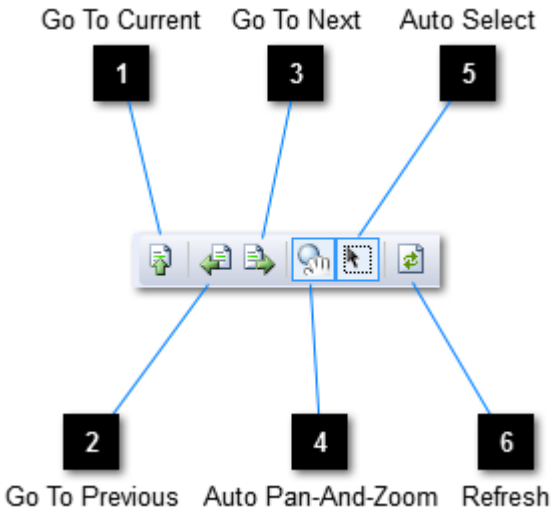
3 Type

The type of the associated matched item.

4 Description

A description of the associated matched item.

Find Results toolbar



1 Go To Current

Actions the currently selected item in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

2 Go To Previous

Selects the previous item in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

3 Go To Next

Selects the next item in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

4 Auto Pan-And-Zoom

When selected, causes the actioned item to be brought into view and scaled such that it fills the viewport.

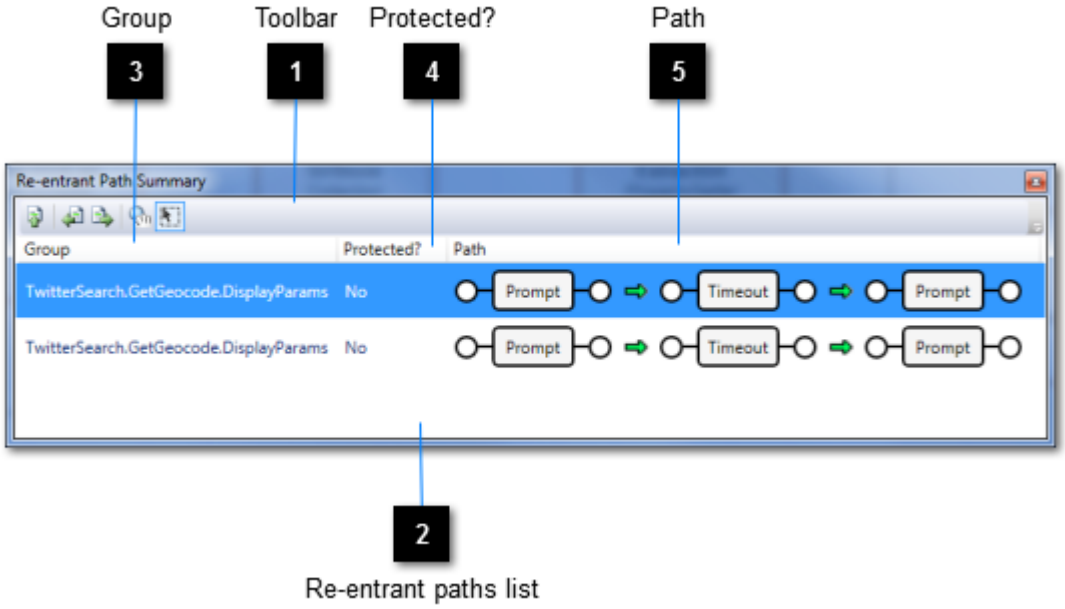
5 Auto Select

When selected, causes the actioned item to be selected.

6 Refresh

Performs the last find operation again and refreshes the list with the new matched items.

Re-entrant Path Summary



Re-entrant paths are described here. The Re-entrant Path Summary is shown by selecting **View → Re-entrant Path Summary** from the menu bar or by clicking on the relevant link in the Status bar. Note that if there is no such link in the Status bar then that is an indication that your project has no re-entrant paths.

1 Toolbar

The toolbar supports actions in respect of the re-entrant paths list.

2 Re-entrant paths list

The list of detected re-entrant paths.

3 Group

The fully-qualified name of the feature group that contains the associated re-entrant path.

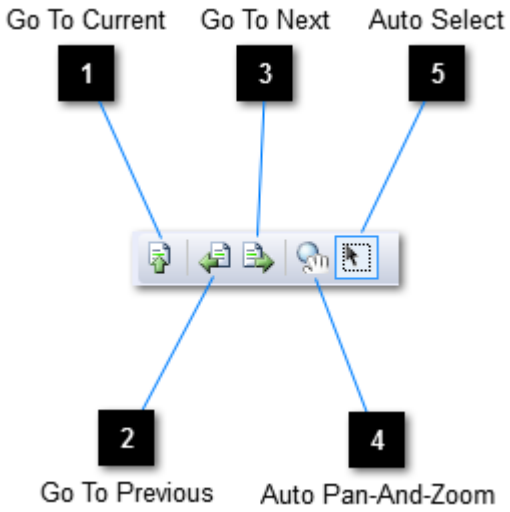
4 Protected?

Indicates whether the associated feature group is a password-protected project. Any such project is always listed, regardless of whether it actually contains any re-entrant paths, merely to warn that there may be re-entrant paths hidden from you. The Editor cannot reveal the contents of secure projects.

5 **Path**

For projects that are not protected, this field shows the chain of features and their interconnections that collectively represent a re-entrant path.

Re-entrant Path Summary toolbar



1 Go To Current

 Actions the currently selected path in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

2 Go To Previous

 Selects the previous path in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

3 Go To Next

 Selects the next path in the list and actions it in accordance with the selections for Auto Pan-And-Zoom and Auto Select.

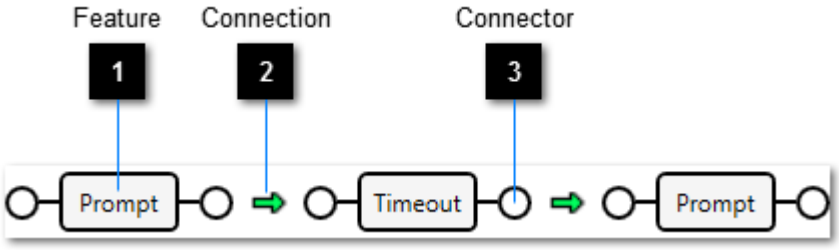
4 Auto Pan-And-Zoom

 When selected, causes the actioned path to be brought into view and scaled such that it fills the viewport.

5 Auto Select

 When selected, causes the actioned path to be selected.

Re-entrant path detail



The above shows an example of the chain of features and their interconnections that collectively represent a re-entrant path. Note that an overlapping path is always shown, meaning that the feature and connectors at the start and end of the chain are the same.

1 Feature

Prompt Represents a feature in the re-entrant path. Click it to action the feature.

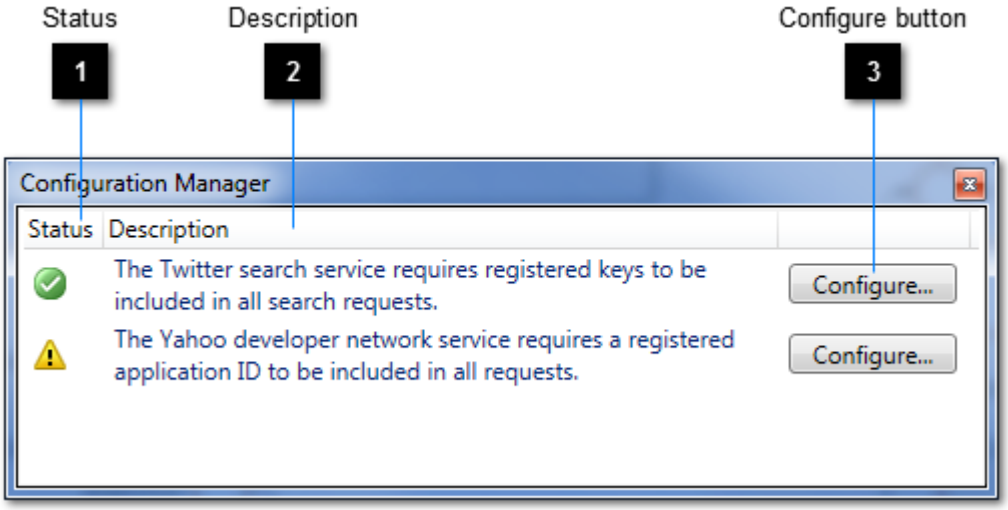
2 Connection

Represents a connection in the re-entrant path. Click it to action the connection.

3 Connector

Represents a connector in the re-entrant path. Hover the mouse over it to show a tool-tip containing its user-assigned name. Click it to action the connector.

Configuration Manager



The Configuration Manager identifies services that need information from you before they can operate correctly. For any given service you will only need to enter the information once because it is cached on your computer under your account. Note that the Editor examines the current project to determine whether any configurable services will be required and, if at least one such service exists that has yet to be configured, you will be prompted via a link in the Status bar.

The Configuration Manager is shown by selecting **View → Configuration Manager** from the menu bar or by clicking on the aforementioned Status bar link. Note that if there is no such link then that is an indication that no services require configuring.

1 Status

Provides a visual indication as to the status of the configurable service. Those with an amber warning triangle require you to supply information.

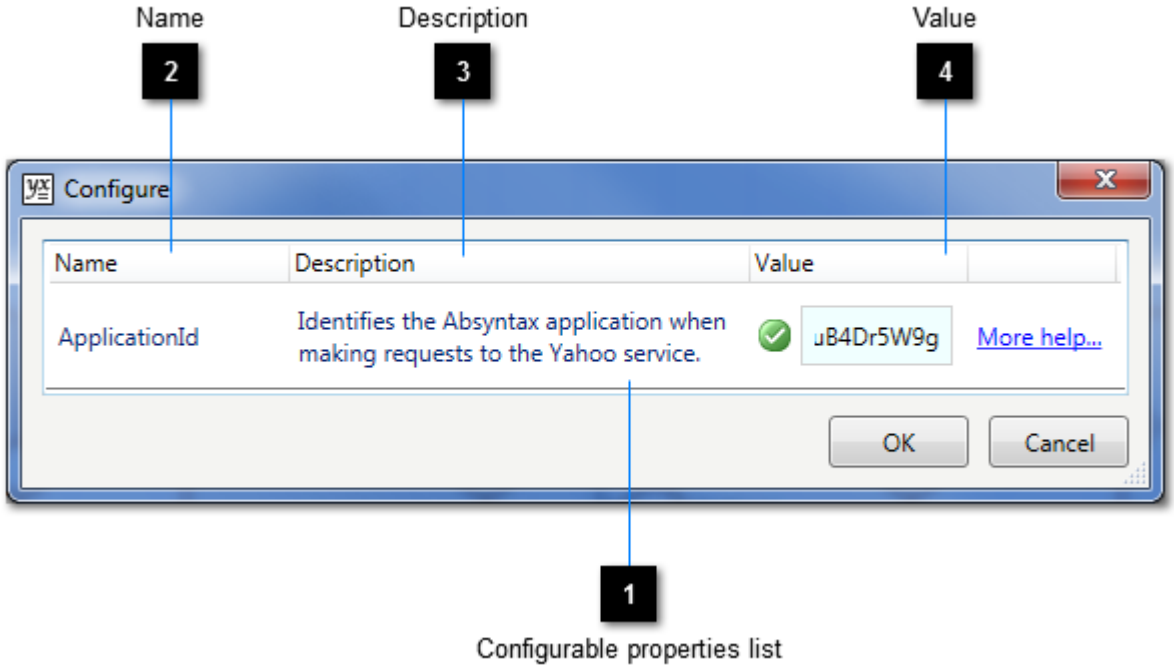
2 Description

A description of the configurable service.

3 Configure button

Press this button to open the Configure dialogue.

Configure



1 Configurable properties list

A list of one or more configurable service properties that require you to specify a value.

2 Name

The name of a property for which you must supply a value.

3 Description

The description of a property for which you must supply a value.

4 Value

The supplied value. A visual cue is provided to indicate whether the specified value is or appears to be valid.

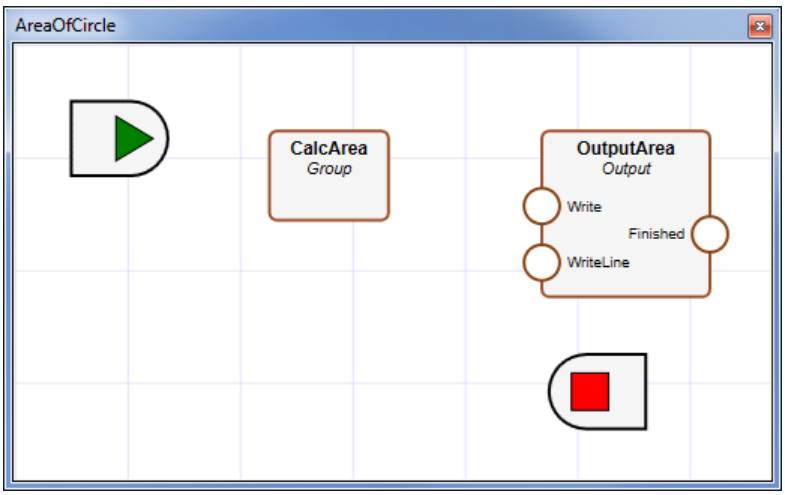
Connector Tray

The Connector Tray is used in conjunction with configurable feature group views. Whenever you want to add an input or output to a configurable feature group, click and drag the relevant connector template onto the design surface. If you have chosen a data connector then the Type Browser will open, prompting you to select the type of data that the connector is to transmit. Input connectors transmit signals and data from outside the group to the group's contained features; output connectors transmit signals and data from within the group to features that are connected to the group. You could think of these connectors as channels of signals and data operating across the group's boundary.

The result of such an operation is that a connector is added to the connector profile of the containing feature group. Furthermore, a proxy for this connector appears on the design surface and it this proxy that enables you to connect the group's contained features to the group's sibling features. Also, by opening the view of the group's parent you can see the group's added connectors and use them to make connections with other features.

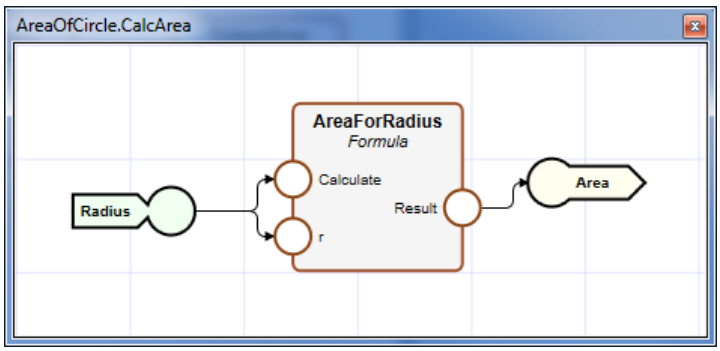
The Connector Tray is shown by selecting **View → Connector Tray** from the menu bar.

Below is an example of a project, "AreaOfCircle", that contains two features. One of them, "CalcArea", is a **Group** – a feature representing a configurable feature group. Currently CalcArea has neither inputs nor outputs and in this form it serves no purpose.

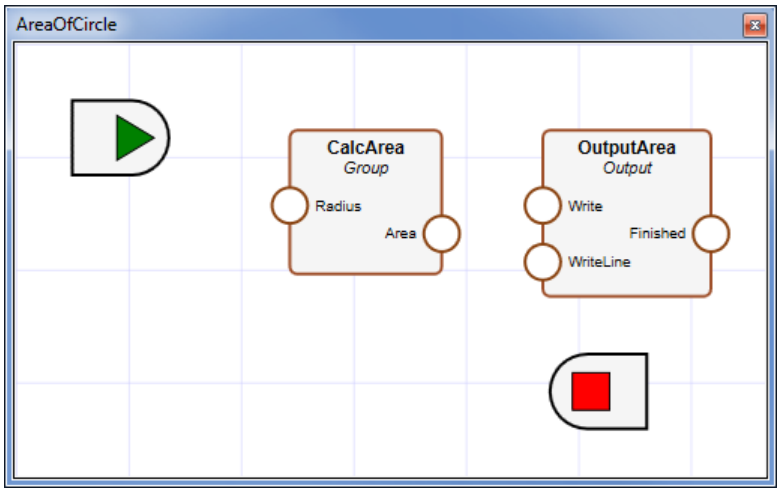


Below is a view of the CalcArea group. Connectors named "Radius" and "Area" have been added using the technique described above and wired up to a **Formula** feature whose job is to perform some calculation. Looking at this view you might reasonably infer the following sequence of events:

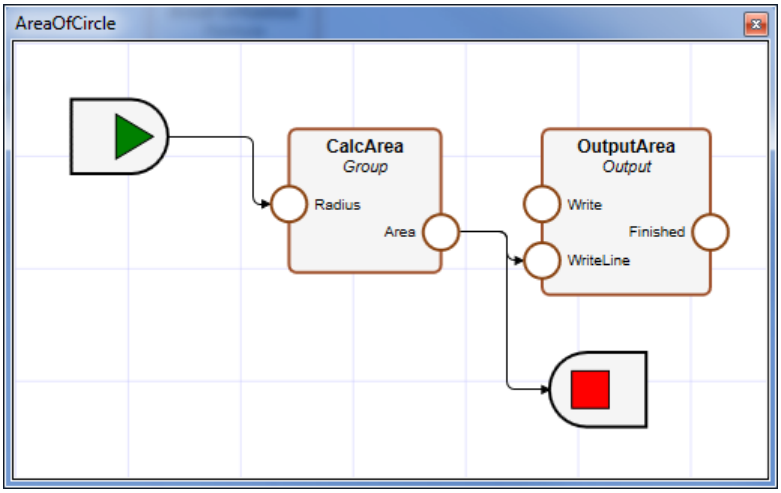
1. a value representing a circle's radius is received by the group;
2. this value is used to set the **Formula**'s input parameter, "r", and then invoke the calculation;
3. the result of the calculation is passed out of the group.



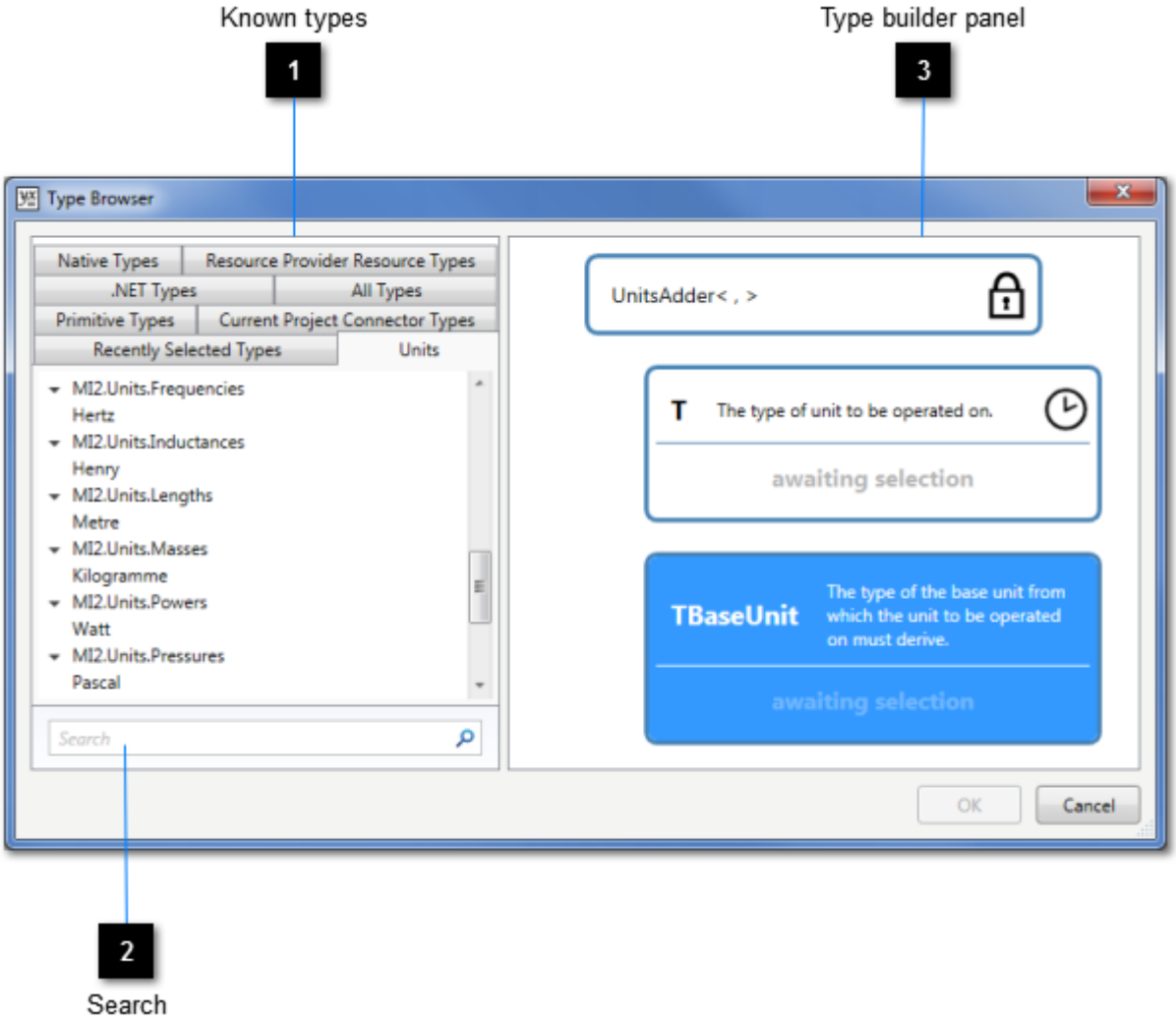
The effect of adding the aforementioned connectors to the CalcArea group can be seen in the parent view. The group is now ready to be wired up.



Below is a view of the finished project. The CalcArea group thus represents an operation, the details of which are hidden at this level. Furthermore, the operation supports clearly defined inputs and outputs. All of which serves to demonstrate one of the key benefits of configurable feature groups.


AreaOfCircle
CalcArea
Group
Radius
Area
OutputArea
Output
Write
Finished
WriteLine

Type Browser



The Type Browser dialogue is opened whenever the Editor needs you to select a type. The dialogue consists of two panes. The left-hand pane contains categorised lists of all types known to the Editor, along with a search box for narrowing down a type search. The right-hand pane contains the selected type, rendered as a hierarchical tree. All nodes in this tree that are annotated with the words "awaiting selection" require you to choose suitable types, and you will not be allowed to proceed until you have resolved all of them. Select each such node in turn, then search for and choose the required type.

Under certain circumstances, as well as being able to select types from the Type Browser's known type lists you can also select types using the connector data type sampler tool. This tool allows you to browse the current project and "pick" a type from any connector therein that supports the transmission of data. When the dropper turns green, the connector over which the cursor is positioned may have its type sampled by clicking on it. When the dropper turns red, either the connector does not support data or its data type is not eligible for selection.

1 Known types

A collection of categorised lists of all known types.

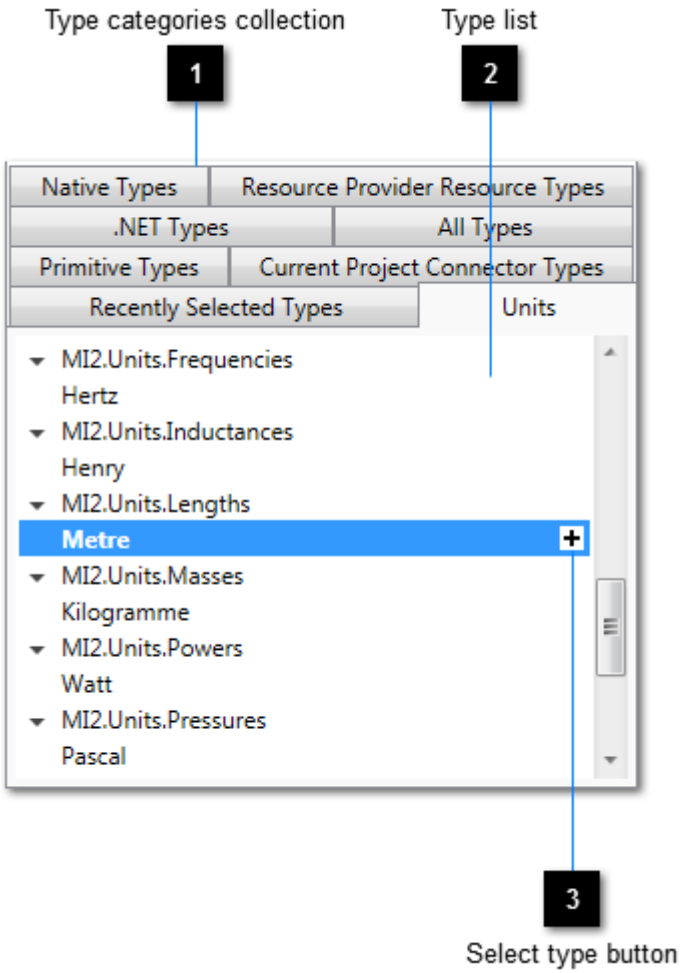
2 Search

Find types using the search facility at the bottom of the dialogue. Words entered here are matched with a type's namespace and name. The filtered results are displayed in the "known types" lists above. (Available types are also filtered according to any constraints associated with the node selected in the type builder panel.)

3 Type builder panel

The type to be returned by the dialogue is built here.

Known type lists



The known type lists form part of the Type Browser dialogue.

1 Type categories collection

A tab collection, each tab representing a categorisation of types. A type may appear in multiple categories. Notable categories include:

Primitive Types	The most commonly used types, including strings and number types.
Current Project Connector Types	The set of unique types supported by data connectors in the current project.
Native Types	Types defined by the various software libraries of the Absyntax framework.

2 Type list

A list of types for the selected category, qualified and ordered by namespace.

3 **Select type button**

 Click this button to specify the associated type as the type for the selected node in the type builder panel. Alternatively, double-click any type in the type list.

Type builder panel

The type builder panel forms part of the Type Browser dialogue. It presents a root type that must be resolved. To achieve this, types are found and selected using the known type lists. Sometimes a single type will be all that you require. However, more complex root types can be built and these are presented in the panel as a type node hierarchy, an example of which is shown above. The description associated with each node will help you to understand what it is you are being asked to select. Additionally, not all types will necessarily be eligible for selection for a given node. So when you select a node, the types available for selection may be a subset of the known types.

1 Resolved type name

The name of the fully resolved type.

2 Locked symbol

Indicates that the type associated with the node cannot be changed.

3 Pending symbol

Indicates that the type associated with the node cannot be selected until other nodes' types have been selected.

4 Description

A description of the type requiring selection.

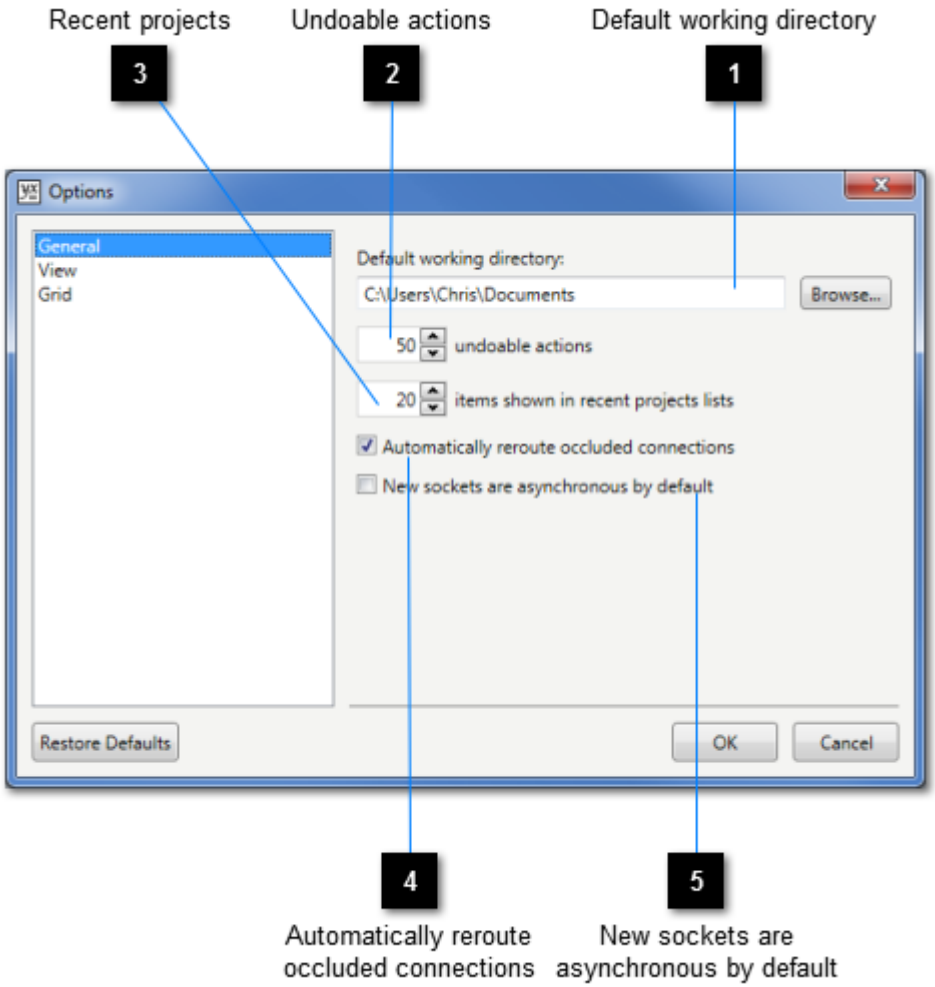
5 Awaiting selection indicator

Indicates that a type has yet to be selected. When a type has been selected, its name is displayed here instead.

Options

You can specify your Editor preferences via the Options dialogue, which is shown by selecting **Tools → Options...** from the menu bar. Any values you set are stored and used for subsequent sessions.

Options - General



1 Default working directory

This option specifies the path of the directory from which projects are opened and to which they are saved by default.

2 Undoable actions

This option specifies the maximum number of undoable actions that the Editor is to cache. The higher the number, the more resources the Editor will require of your computer.

3 Recent projects

This option specifies the maximum number of projects to be presented in the Recent Projects list (select **File → Recent Projects** from the menu bar).

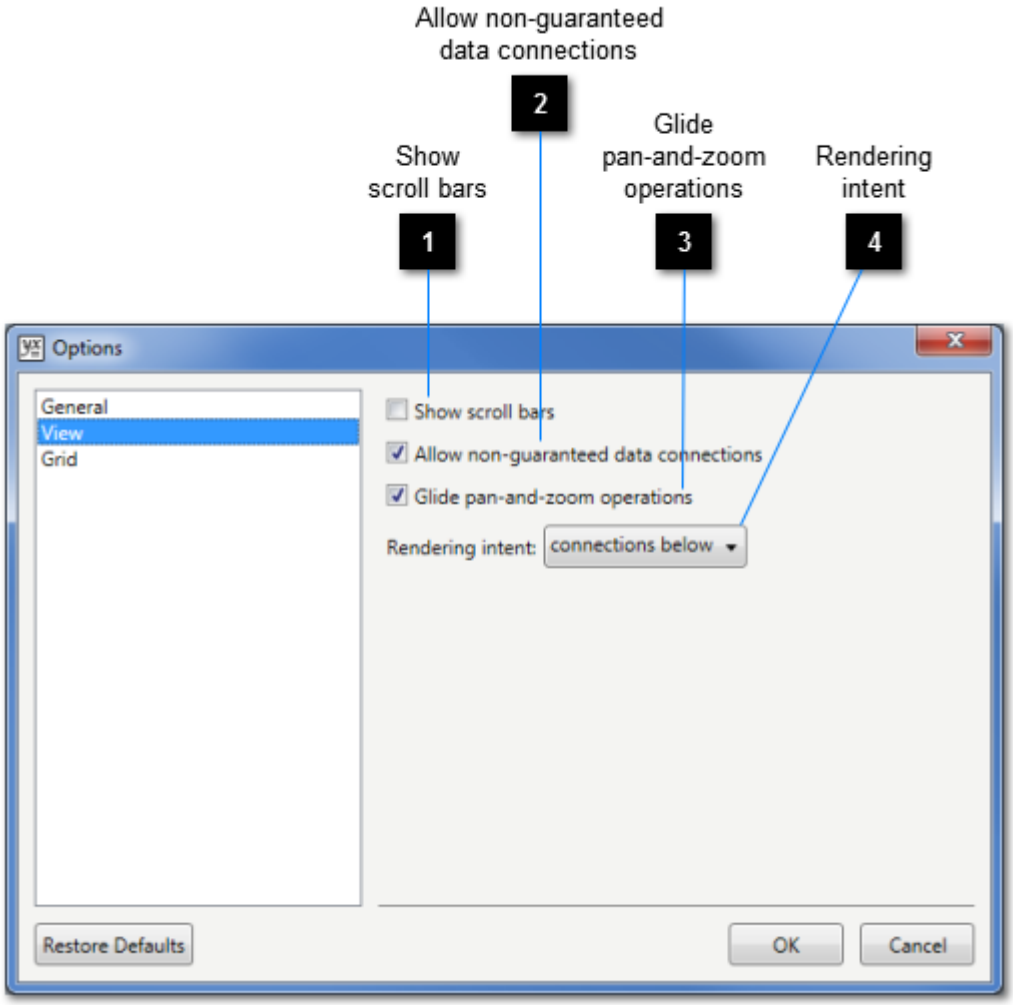
4 **Automatically reroute occluded connections**

When a feature is added or moved on top of existing connection paths on the design surface, this option determines whether the Editor reroutes the affected connections.

5 **New sockets are asynchronous by default**

This option determines whether newly created sockets are flagged as asynchronous by default.

Options - View



1 Show scroll bars

Select this option if you want scroll bars to appear in all views.

2 Allow non-guaranteed data connections

Select this option if you want the Connection Tool and the Connection Manager to allow connections to be created between data connectors for which a non-guaranteed data conversion would be required.

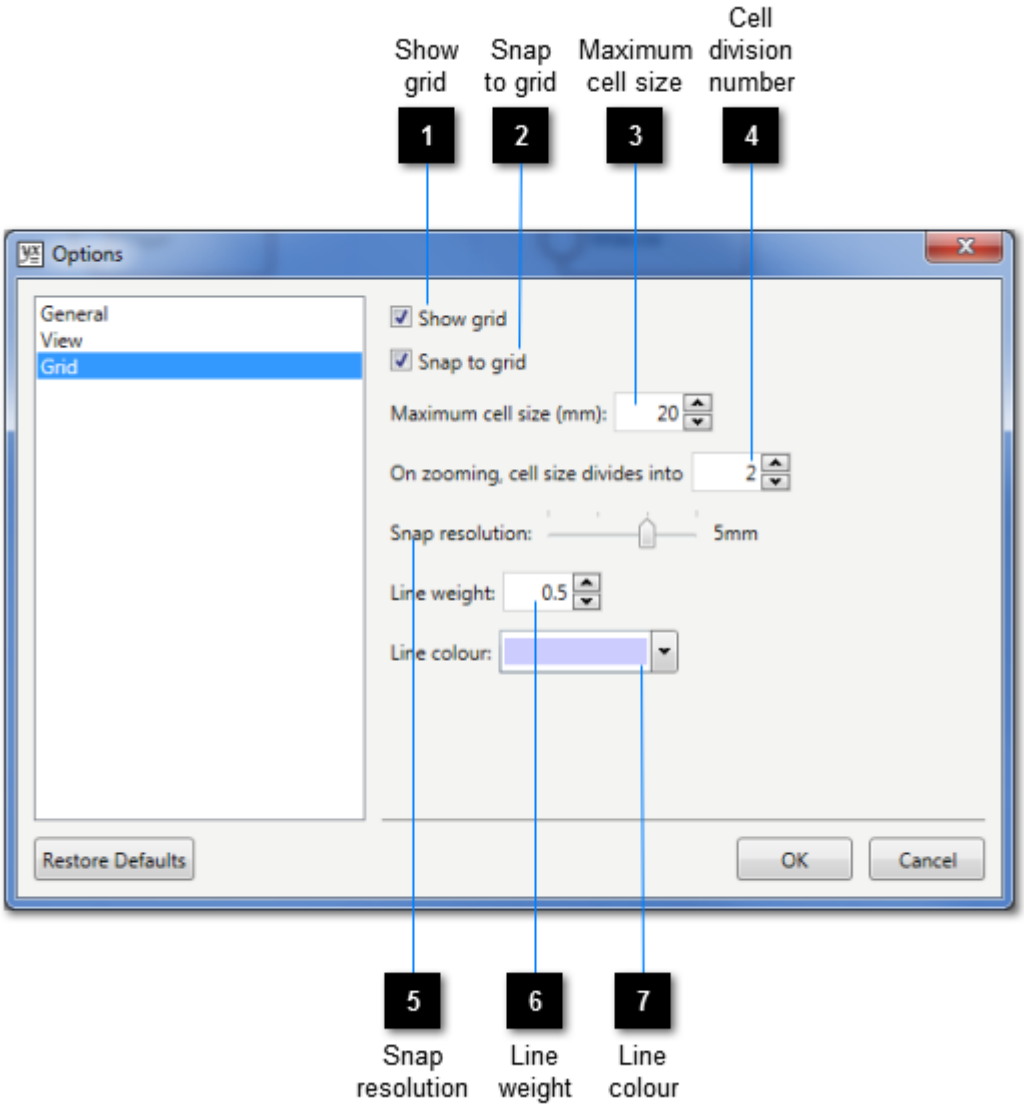
3 Glide pan-and-zoom operations

Select this option if pan-and-zoom operations are to be animated.

Rendering intent

This option determines whether connections are rendered above or below features in a view.

Options - Grid



1 Show grid

Check this option if you want a grid to be visible on a view's design surface.

2 Snap to grid

Check this option if you want shapes to be snapped to a grid (visible or otherwise) on a view's design surface.

3 **Maximum cell size**

Grid cells resize themselves to counter the effects of zooming. This option specifies the maximum size (in millimetres) that a grid cell can reach before it is subdivided.

4 **Cell division number**

This option specifies the number of cells horizontally and vertically that a cell divides into when its maximum size is reached.

5 **Snap resolution**

This option defines the distance between a grid's snap points.

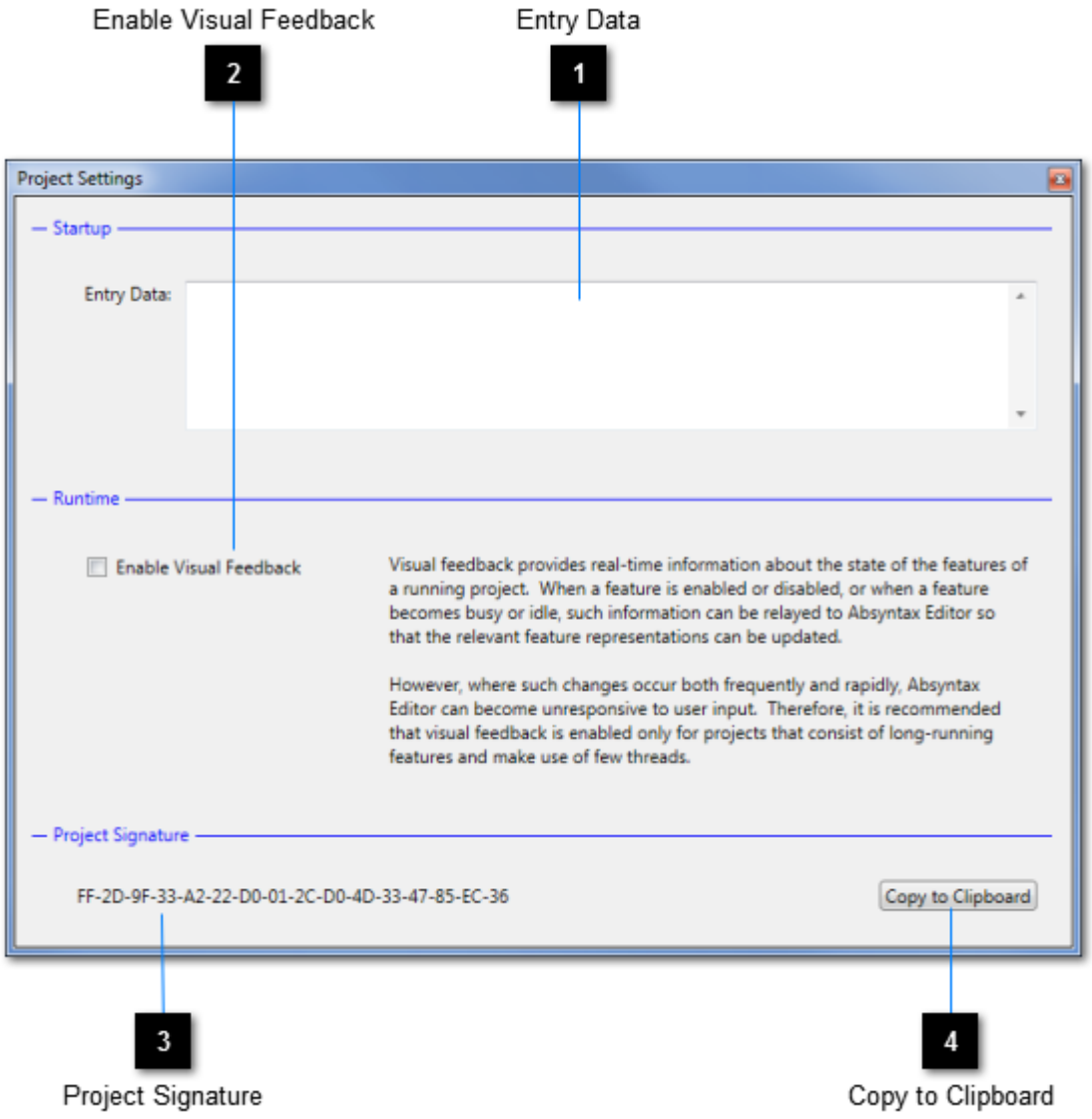
6 **Line weight**

This option specifies the thickness of the lines used to render a grid.

7 **Line colour**

This option specifies the colour to be used to render grid lines.

Project Settings



Settings can be obtained for and applied to the current project. The Project Settings page is shown by selecting **View** → **Project Settings** from the menu bar.

1 Entry Data

This field is enabled if your project's entry-point requires data. Refer to the rules governing the specification of values in the Absyntax Batch Client's data option for more details.

2 Enable Visual Feedback

Some features can take a relatively long time to complete their operations. While executing a project in the Editor, you may want to receive visual cues indicating whether such features are busy, in which you case you should select this option.

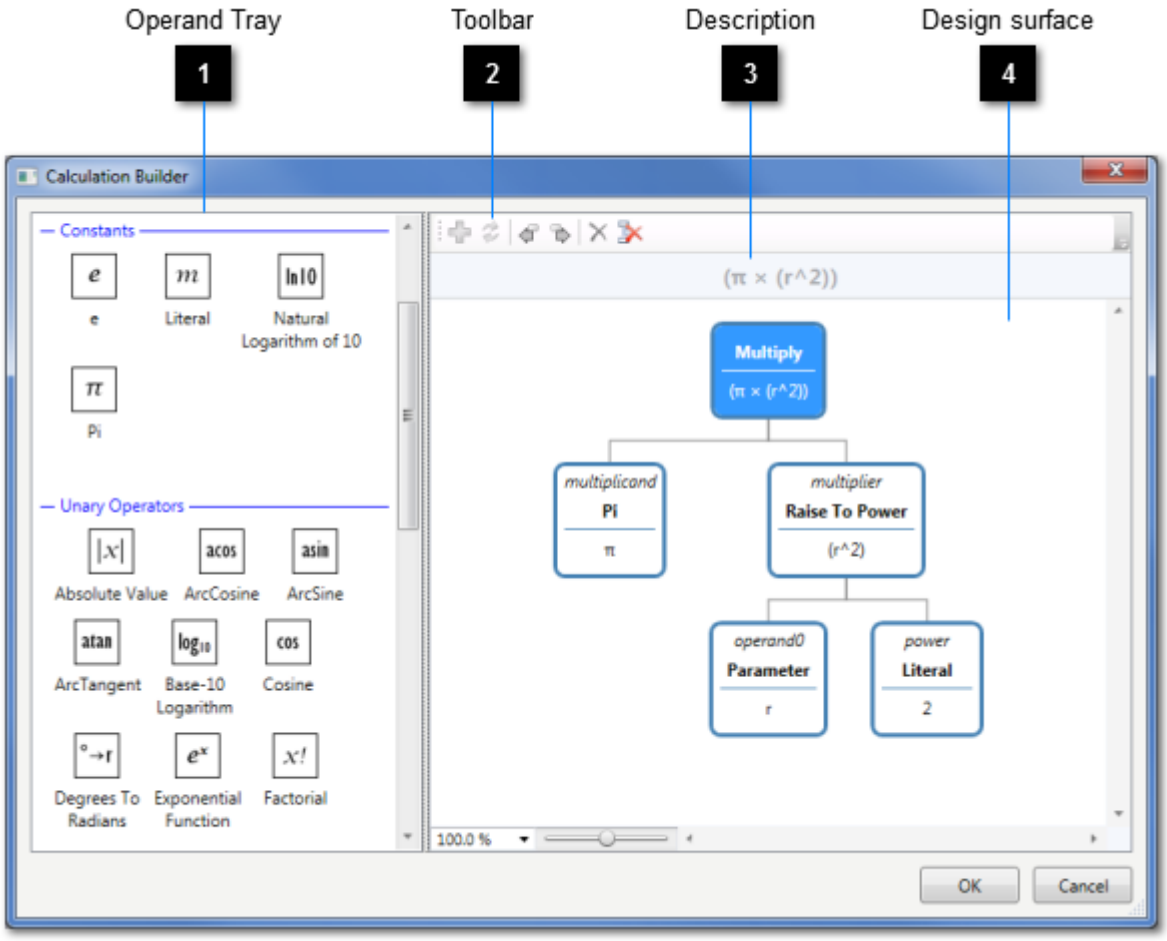
3 Project Signature

This field, which is evaluated every time the page is activated, shows the current project's signature.

4 Copy to Clipboard

Click this button to copy the project signature to the clipboard, making it available for use in other contexts (such as in conjunction with the Absyntax Batch Client).

Calculation Builder



The Calculation Builder dialogue is most commonly used in conjunction with the **Formula** feature. It facilitates the creation and maintenance of mathematical expressions without the need to understand any proprietary syntax, operator precedence, use of parentheses and the like.

1 Operand Tray

The list of available operators and operands that may be combined on the design surface when building a mathematical expression.

2 Toolbar

The toolbar supports various design surface actions.

3 Description

A representation of the current mathematical expression.

4 Design surface

The design surface on which you build your mathematical expressions.

Mathematical Expressions

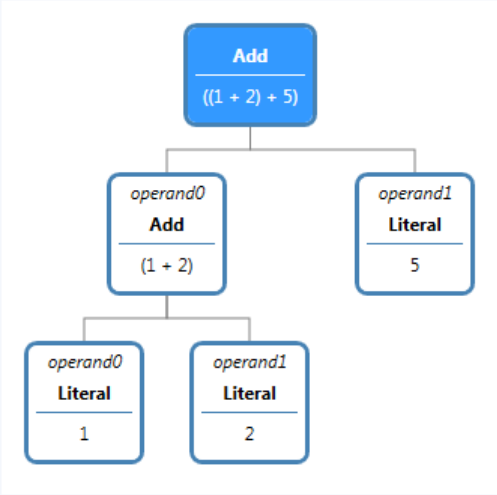
Mathematical expressions take the form of a hierarchy of operators and operands. An operator is a mathematical operation that uses one or more operands to calculate a value. An operand is a value on which a mathematical operation is performed. Importantly, an operand can itself be an operator, meaning that its value must be calculated.

For example, the following expression has two "add" operators and three constant values (1, 2 and 5):

$$y = 1 + 2 + 5$$

The first operator adds 1 and 2, then passes the result (3) to the second operator, whose job is then to add 3 and 5.

In the Calculation Builder, such an expression would look like this:



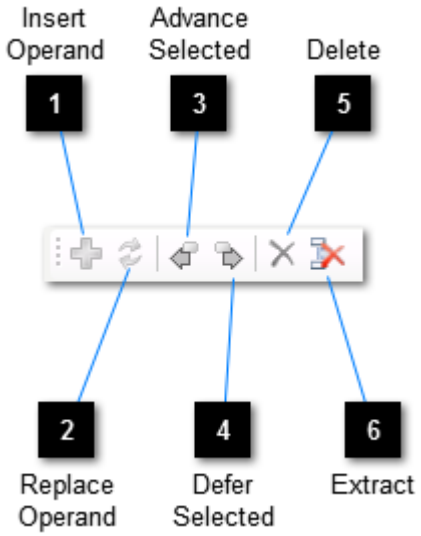
In Absyntax there are two kinds of operators: *unary* operators and *binary* operators. Unary operators have just one operand. An example is the **Square Root** operator. Binary operators have two operands and the **Add** operator is such an example.

An operand of high importance is the **Parameter**. Parameters represent inputs to an expression that must be supplied before the expression can be evaluated. You assign a name to each **Parameter**. Multiple parameters may appear in the node hierarchy and parameters with the same name can also be used. Where multiple **Parameter** nodes share the same name, they will be assigned the same value when the expression is evaluated.

Finally, there is a group of operands whose values are fixed. Such operands are known as *constants*. The values of most of these operands are defined by Absyntax; the **Literal** operand, though, yields a value that is defined by you.

The root node in the hierarchy embodies the overall expression. When an expression is evaluated, the root node is evaluated first. If it is an operator then each of its operands is evaluated before the operation defined by the operator can be performed. This rule is used to resolve each and every node in the hierarchy.

Calculation Builder toolbar



1 Insert Operand

Inserts the selected operator from the Operand Tray at the position of the currently selected node on the design surface, at the same time making the selected node an operand of the newly inserted operator.

The same effect can be achieved by dragging an item from the Operand Tray onto the node above which the new item is to be inserted, while at the same time holding down the Ctrl key.

2 Replace Operand

Replaces the currently selected node on the design surface with the selected item in the Operand Tray. Any existing operands of the replaced node are used as operands for the new item. If the new item is not an operator (i.e. it does not support operands) then the existing operands will be discarded.

The same effect can be achieved by dragging an item from the Operand Tray onto the node to be replaced.

3 Advance Selected

Swaps the selected design surface node with its previous sibling.

4 Defer Selected

 Swaps the selected design surface node with its next sibling.

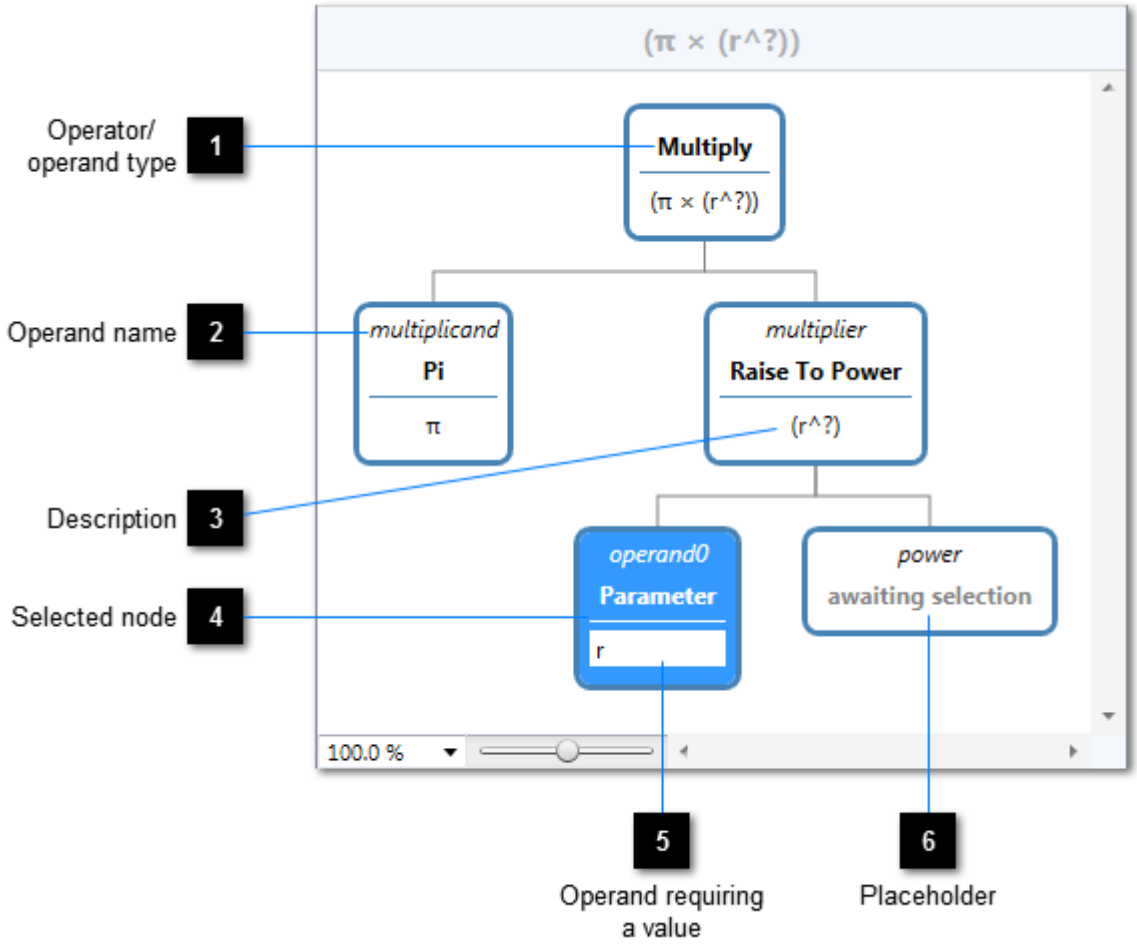
5 **Delete**

 Deletes the currently selected design surface node and all its operands if it has any. The same effect can be achieved by pressing Delete.

6 **Extract**

 Deletes the currently selected design surface operator node and replaces it in the node hierarchy with its first operand.

Anatomy of a mathematical expression



1 Operator/ operand type

The type of the operator or operand.

2 Operand name

The name of the operand in the context of its operator. In this example, the first operand of a **Multiply** operator is known as the multiplicand.

3 Description

A description of the associated node. In the case of operators, the description is based on the descriptions of its operands. The presence of a question mark indicates that an operator placeholder has yet to be filled. In such instances the expression is deemed to be incomplete.

4 Selected node

The selected node is the target of Calculation Builder operations.

5 Operand requiring a value

Some operands – specifically the **Parameter** and the **Literal** – require you to enter a value. The **Parameter** needs a name (in this example a **Parameter** has been assigned the name *r*, perhaps to represent the radius of a circle) and the **Literal** requires a numeric value. To specify a value for such operands, select the node. This will display a value editor: use this to change the operand value.

6 Placeholder

A placeholder is an operand that has yet to be selected. It is denoted by the words "awaiting selection".

Batch Client

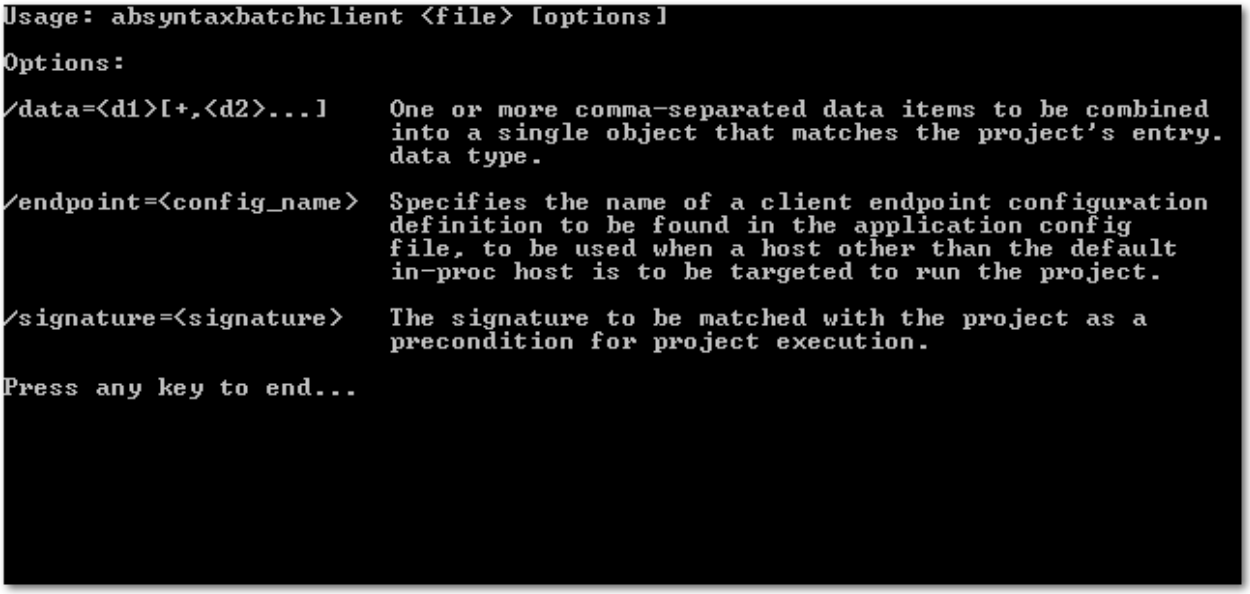
The Absyntax Batch Client is a program whose purpose is to execute Absyntax projects without the interactive runtime capabilities of the Absyntax Editor. The program can be found in the Absyntax installation directory and is named AbsyntaxBatchClient.exe.

Note that the Batch Client offers no means of pausing or resuming a project during execution and so ignores any project breakpoints that may exist. You can abort a running project, though, by pressing Return or Enter. Otherwise, the project will continue to execute until it outputs a signal or data through its exit-point.

The Batch Client is the component to use to execute standalone projects that no longer require development. It requires fewer of your computer's resources and performs much faster than the Editor.

Running the program

Find the Batch Client program, AbsyntaxBatchClient.exe, by navigating to the Absyntax installation directory using your preferred method. (In Windows, use either a command line window or the Windows/File Explorer.) Then run it: you will see the following window.



This window displays the Batch Client's command-line usage options for executing an Absyntax project. As a minimum you must specify the full path of the file containing the project to be executed. For example:

```
AbsyntaxBatchClient.exe C:\Projects\Fibonacci.apj
```

Data

For projects whose Entry input requires data you must qualify the command with the `/data` option. For example:

```
AbsyntaxBatchClient.exe C:\Projects\TwitterSearch.apj /data=Madrid
```

Note that this will only work if the data ("Madrid", in this case) can be converted by Absyntax from a string into the type of data required by the project's entry-point. Here's another example, showing one way of supplying multiple data items.

```
AbsyntaxBatchClient.exe C:\Projects\Sort.apj /data=46,-1,999
```

For the final word on data formatting options, see here.

Endpoint

By default, the Batch Client delegates the task of project execution to an in-process service that it creates itself. However, it is possible to redirect the Batch Client to use an external service instead, typically on another computer, and this is achieved by providing a value for the `/endpoint` command line option. (Note that such external services are usually hosted by the Absyntax Runtime Server.)

The value assigned by this option is the name of an endpoint whose characteristics are defined in the configuration file, "AbsyntaxBatchClient.exe.config". An endpoint encapsulates the details necessary for the Batch Client to communicate with the service. In order to use this option effectively you should have a working knowledge of Windows Communication Foundation (WCF).

The section of relevance in the configuration file is shown below.

```
<configuration>
        <system.serviceModel>
                <client>
                        <endpoint name="NetNamedPipeBinding_IBasicProjectService" ... />
                        <endpoint name="NetTcpBinding_IBasicProjectService" ... />
                </client>
        </system.serviceModel>
</configuration>
```

Typically you would create a client endpoint in this section, with an address, binding and contract to match those exposed by the target service.

Signature

If you need to ensure that a project should be executed only if its signature matches a known value, you include a value for the `/signature` command line option, an example of which follows.

```
AbsyntaxBatchClient.exe C:\Projects\Fibonacci.apj /
signature=2F-40-F2-BA-68-02-8D-30-78-FA-85-82-97-51-8C-84
```

The value you use for the signature can be obtained via the Absyntax Editor's Project Settings page.

Command line data

For the majority of scenarios, specifying project start-up data on the command line is straightforward.

```
AbsyntaxBatchClient.exe C:\Projects\TwitterSearch.apj /data=Madrid
```

It is possible that a project requires multiple items of data to be supplied to its entry-point, in which case commas should be used to separate individual data items. The following example shows how to supply data to a project requiring an array of integers at start-up.

```
AbsyntaxBatchClient.exe C:\Projects\Sort.apj /data=46,-1,999
```

In this case the string "46,-1,999" is passed to the Batch Client, where it is split into substrings representing the individual data items and converted to the data type required by the target project.

The rules for defining command-line data items are as follows.

1. Commas must be used to separate substrings.
2. Where a comma is to be included in a substring, enclose it in a pair of double-quotes.
3. Where a double-quote is to be included in a substring, it must not abut against a comma that acts as a separator. Thus, to ensure that double-quote pairs do not appear in a substring they must both abut against either the start/end of the string or a comma therein that is not itself enclosed in double-quotes. The lack of a terminating double-quote will result in the last substring terminating with the last character in the string.
4. Null values are denoted by abutting pairs of commas.
5. Empty strings are denoted by abutting pairs of double-quotes that both abut against commas acting as separators or the start/end of the string.
6. White space is not trimmed.

Here are some examples:

Command Line Option	Substring Count	Substrings	Description
/Data=	1	[NULL]	One null value.
/Data=\"\"	1	[""]	One empty string.
/Data=1	1	["1"]	
/Data=,	2	[NULL,NULL]	Two null values.
/Data=,,	3	[NULL,NULL,NULL]	
/Data=, ,	3	[NULL," ",NULL]	
/Data=,\"\",	3	[NULL,"",NULL]	
/Data=,\"\"	2	[NULL,""]	
/Data=\"	1	["""]	One double-quote character.
/Data=1,\"2,3\"	2	["1","2,3"]	
/Data=\"1,2	1	[""1,2"]	

/Data=\"1,2\",3	2	["1,2","3"]	
/Data=The,quick,brown,fox	4	["The","quick","brown","fox"]	
/Data=The quick brown fox	1	["The quick brown fox"]	
/Data=The quick,brown fox	2	["The quick","brown fox"]	
/Data=The quick, brown fox	2	["The quick"," brown fox"]	

Executing projects

Executing signal-input projects

Signal-input projects have entry-points that do not require data. Such projects can be executed by the Batch Client in several ways. The techniques described below all work because, during installation, a default action was associated with .apj files to invoke the Batch Client on the target file.

Command Line Window

1. Open a command line window.
2. Type the path and file name. For example, "`C:\Projects\Fibonacci.apj`".

Windows/File Explorer

1. Navigate to the project.
2. Double-click on it.

Batch File

1. Create a text file with a .bat extension.
2. Type into it the path and file name. For example, "`C:\Projects\Fibonacci.apj`".
3. Save the file.
4. Run the batch file by invoking it from a command line window or double-clicking it in Windows/File Explorer.

Executing data-input projects

Data-input projects have entry-points that require data. Such projects can be executed by the Batch Client in the following ways.

Batch File

1. Create a text file with a .bat extension.
2. Type into it the path and file name of the Batch Client program, followed by all necessary arguments. For example, "`C:\Program Files (x86)\MII Ltd\Absyntax Framework\AbsyntaxBatchClient.exe C:\Projects\TwitterSearch.apj /data=Madrid`".
3. Save the file.
4. Run the batch file by invoking it from a command line window or double-clicking it in Windows/File Explorer.

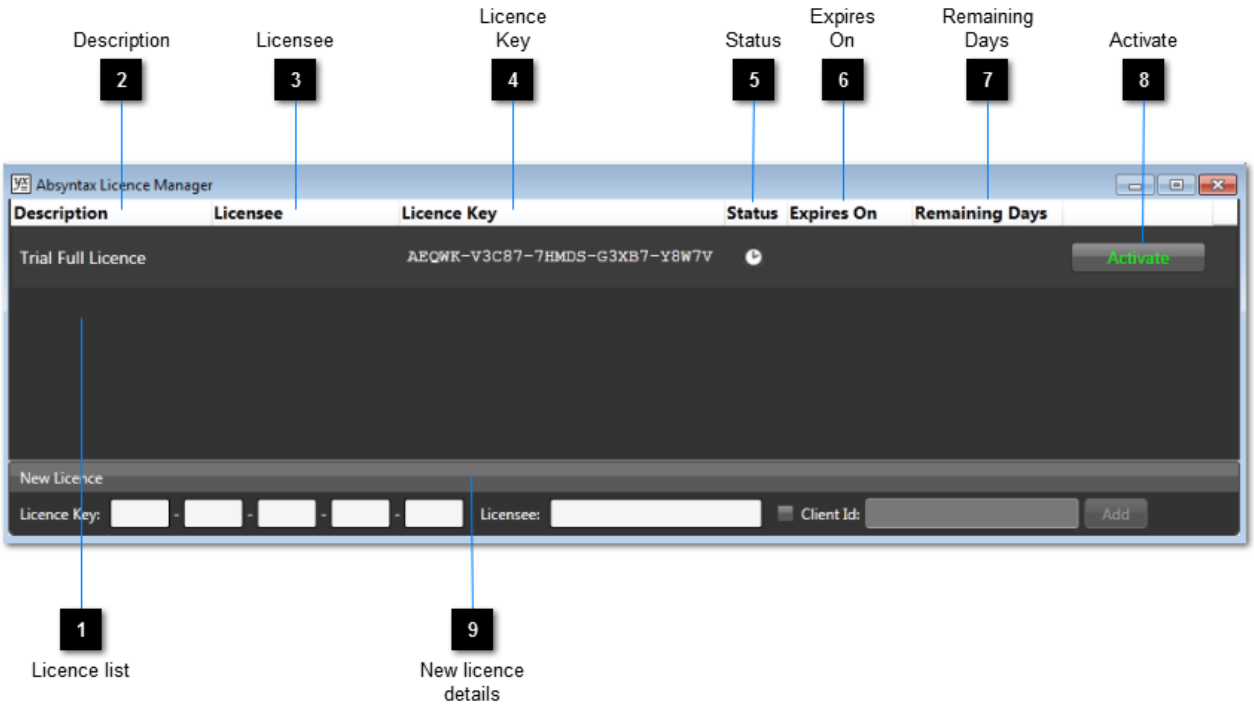
Command Line Window

1. Open a command line window.
2. Type the path and file name of the Batch Client program, followed by all necessary arguments. For example, "`C:\Program Files (x86)\MII Ltd\Absyntax Framework\AbsyntaxBatchClient.exe C:\Projects\TwitterSearch.apj /data=Madrid`".

Runtime Server

The Absyntax Runtime Server is a program whose purpose is to host multiple executing Absyntax projects and, more commonly, remote feature groups concurrently. The Runtime Server is used by the Absyntax Editor to execute projects in an isolated environment. You may also use it to facilitate parallel processing scenarios.

Licensing Manager



Use the Absyntax Licensing Manager to add, activate and monitor your licences for the Absyntax Framework. It can be started by double-clicking the desktop shortcut that was created during installation or by navigating to the installation directory and invoking AbsyntaxLicenceManager.exe.

1 Licence list

The list of all added licences. Absyntax installs with a trial licence that provides you with full, unrestricted access to all aspects of the Framework.

2 Description

A description of the type of the licence.

3 Licensee

The name of the user or company to whom the licence is granted. Trial licences do not have a licensee.

4 Licence Key

An alphanumeric string encapsulating details of the licence.

5 Status

A visual cue as to the state of the licence. Options are:

- Pending
- Activated
- Expired
- Invalid

6 Expires On

The date on which the licence expires.

7 Remaining Days

The number of days remaining before the licence expires.

8 Activate

Click this button in order to activate a licence. You will need an internet connection for this to work. You can activate a licence at any time and you can have multiple activated licences. Absyntax compounds the permissions of all active licences in order to determine the extent to which you can use the Framework and its components. Note that a licence's expiry date is determined once you have activated it.

9 New licence details

To enter the details of a new licence, complete the five licence key fields and the licensee field. If you are entering a key for a third-party execution licence you must additionally check the Client Id check box and enter the unique client identifier that accompanied your licence key. Once you have entered all necessary information, the Add button will become enabled: click this to add the licence to the list. You will be advised of any incorrect details, so be careful to enter all information accurately. The value you enter for the licensee field – which must be the same as the value you provided when purchasing the licence – will be displayed in the Absyntax Editor.

FAQs

General

This section contains frequently asked questions pertaining to the Absyntax Framework in general.

What is a string?

A string – or, more specifically, a *character* string – is a programming term used to describe a sequential collection of zero or more alphanumeric and symbolic characters. Strings are often used to store and display text.

What is a Boolean expression?

A Boolean expression is a combination of constant values, variables, operators and functions that, when evaluated, produces a Boolean value (i.e. either "true" or "false"). An example of a Boolean expression is as follows:

$$x > 1$$

This expression represents a single test on a given number, x. It yields true for all values of x greater than 1 and false for all values of x less than or equal to 1.

A more complex example, involving two tests, is shown below:

$$x \geqslant 1 \text{ AND } x \leqslant 10$$

This expression, which makes use of the Boolean "AND" operator, yields true only if the value of x is in the range 1 to 10 inclusive.

Absyntax uses Boolean expressions in a variety of contexts as the basis for tests. You can create such expressions in the Absyntax Editor using the Filter Builder.

Absyntax Editor

This section contains frequently asked questions pertaining to the Absyntax Editor.

How do I secure a project?

Securing a project involves two steps: setting a password and locking the project. The Editor allows you to manage passwords for the current project. Any project, be it the current project or a nested project, can be locked.

Locking a project

1. Open the view that contains the target project. If the target project is the current project then open the project view (select **View → Project View** from the menu bar).
2. Select the Pointer Tool.
3. Select the target project from the contained features. If the target project is the current project then click on the design surface.
4. In the Properties panel, check the Protect property value if it is not already checked.
5. If a password has been set for this project but you have not yet supplied it, you will now be prompted to enter the password. The lock operation will succeed only if you enter the correct password. Failure to do this will leave the Protect property value unchecked.

Setting the current project's password

1. Select **Project → Set Password...** from the menu bar.
2. The Set Password dialogue will open. You will be prompted to enter both a new password and a confirmation of this new password. If a password has been set for the current project, you will be prompted to enter this as well.

Clearing the current project's existing password

1. Select **Project → Set Password...** from the menu bar.
2. The Set Password dialogue will open. You will be prompted to enter both a new password and a confirmation of this new password and you should leave both fields blank. If a password has been set for the current project, you will be prompted to enter this as well.

Can I load a previously saved project into the current project?

Yes. This is referred to as *importing* a project.

1. Open the view representing the group into which you want to import the project.
2. Select **File → Import Project...** from the menu bar.
3. The Open dialogue is displayed. Use it find and select the project you want to import.
4. Click Open. The project is imported into the group.

Note that an imported project is just like any other feature. It has no relation to the saved project file from which it was imported. Any subsequent modifications to the in situ project or its contents are not reflected in the saved version, which remains unaffected. If you need to save such changes back to the original file then follow these steps.

Can I save an embedded project?

Yes. An embedded project is a project feature that exists within the current root project.

1. Open the view representing the group containing the project feature that you want to save.
2. Select the Pointer Tool.
3. Right-click the project feature, which has the effect of selecting it and displaying the Pointer Tool's context menu.
4. From the context menu, select **Save → Selected Project As...**
5. The Save As dialogue is displayed. Use it to complete the operation.