

# Function Creation

Functions within AIDA are created via an [IFunctionFactory](#). Beside the [built-in](#) functions a user can define new functions by either [scripting](#) them on the fly, or by [coding](#) them. User defined functions can be registered with the function factory's catalog; together with the built-in functions they contribute to the pool of [pre-defined functions](#).

## Pre-defined Functions

*Pre-defined functions* are those functions that are registered with the function factory's catalog and that can be created by specifying their name. There are two types of pre-defined functions: the [built-in](#) functions defined by the JAIDA implementation and the ones that have been added to the catalog by the user.

## Built-in Functions

JAIDA currently provides a handful of pre-defined functions. For list of the functions provided by the JAIDA implementation, their definition and the list of the parameter's names please refer to the [JAIDA Release Notes](#).

In the following example we create a gaussian:

```
// The AIDA Factories
IAnalysisFactory af = IAnalysisFactory.create();
ITree = af.createTreeFactory().create();
IFunctionFactory ff = af.createFunctionFactory(tree);

// Create a gaussian
IFunction gauss = ff.createFunctionByName("gauss", "g");
```

## User Registered Functions

Once a user has defined a function it is possible to [register](#) it with the function factory's catalog for a given name. Registered functions can be created by name just like the built-in functions.

## User Defined Functions

A user define new functions by either [scripting](#) them on the fly or by [writing java classes](#).

## Scripted Functions

Through the function factory a user can easily create a function on the fly by providing a string defining the analytical form of the function, its dimension and the list of parameters. As the function's evaluation requires interpreting the provided string, scripted functions are poor in performance. Nevertheless scripting is a quick way to test and troubleshoot a function.

In the following example we create a scripted function that describes a parabola:

```
// The AIDA Factories
IAnalysisFactory af = IAnalysisFactory.create();
ITree = af.createTreeFactory().create();
IFunctionFactory ff = af.createFunctionFactory(tree);

// Create a scripted parabola
IFunction parabola = ff.createFunctionFromScript("parabola", 1, "a*x[0]*x[0]+b*x[0]+c", "a,b,c", "");
```

If needed it is possible to register this function with the function factory's catalog, even if for performance reasons it would be better to actually [code this function](#) before registering it.

## Coded Functions

Functions can be created by directly instantiating a given class. For a function to be used within an AIDA program it has to implement the at least the [IFunction](#) interface.

The IFunction interface is the interface for functions that need to be plotted. For functions that need to be fitted the [IModelFunction](#) interface is required. In our implementation we provide an abstract class ([AbstractIFunction](#)) that implements most of the methods of the above interfaces. By extending the AbstractIFunction the user must provide an implementation for the *double value(double x[])* method that is what defines the function.

Depending on the requirements the user might want to provide an implementation for additional methods:

- If the function has to be used for unbinned fits and speed is an issue the analytical normalization and the gradient should be provided (the corresponding methods are part of the IModelFunction interface). Please refer to the [function normalization](#) session.

- If the function has to be registered with the function factory's catalog an implementation of the *codelet* has to be provided. Please refer to the following section for more information on the [codelet](#).