C#

Arrays and Lists

```
int[] array = new int[] { 1, 2, 3 };
List<int> list = new List<int> { 1, 2, 3 };
```

Lambdas

```
list.Where(item => item == 1)
Action<int> myAct = new Action<int>((myInt) =>
{
  myInt = 2;
});
```

Control Flow

```
for (int i = 0; i <= length; i++)
{
  expression
}
```

```
foreach (var item in collection)
{
  expression
}
```

```
if (statement)
{
  expression
}
else if (statement)
{
  expression
}
else
{
  expression
}
```

```
switch (variable)
{
  case a:
    expression
    break;
  default:
    expression
    break;
}
```

```
while (statement)
{
  expression
}
```

```
try
{
  expression
}
catch (Exception e)
{
  expression
}
finally
{
  expression
}
```

```
do
{
  expression
} while (statement);
```

Synergy .NET

Arrays and Lists

```
data array, [#]int, new int[#] { 1, 2, 3 }
data list, @List<int>, new List<int>() { 1, 2, 3 }
```

Lambdas

```
list.Where(lambda (item) {item == 1})
lambda MyLambda(myInt)
begin
  myInt = 2
end
data myAct, @Action<int>, new Action<int>(MyLambda)
```

Control Flow

```
data i, int
for i from 0 thru length
begin
  expression
end
```

```
data variable, type
foreach variable in collection
begin
  expression
end
```

```
if (statement) then
begin
  expression
end
else if (statement) then
begin
  expression
end
else
begin
  expression
end
```

```
using variable select
  (a),
    begin
      expression
      exit
    end
  (),
    begin
      expression
      exit
    end
end
```

```
while (statement) do
begin
  expression
end
```

```
try
begin
    expression
end
catch (e, @exception)
  begin
    expression
  end
finally
  begin
    expression
  end
endtry
```

```
do statement
begin
  expression
end
until (statement)
```

SYNERGEX

Resource Center: synergex.com/welcomeRC
Documentation: synergex.com/docs

Email: support@synergex.com
Call: 800.366.3472
916.635.7300

	C#	Synergy .NET

Namespace	`namespace MyNamespace`
	`{`
	`  …`
	`}`
		`namespace MyNamespace`
		`  …`
		`endnamespace`

Class & Constructor	`class MyClass {`
	`  MyClass (string args)`
	`  {`
	`  …`
	`  }`
	`}`
		`class MyClass`
		`  method MyClass`
		`  args, string`
		`  endparams`
		`  proc`
		`  …`
		`  endmethod`
		`endclass`

Method	`static int MyMethod(string param)`
	`{`
	`  …`
	`  return 42;`
	`}`
		`static method MyMethod, int`
		`  param, string`
		`  endparams`
		`proc`
		`  mreturn 42`
		`endmethod`

Calling a Method	`int argA = 10;` `MyMethod(argA);`	`data argA, int, 10` `MyMethod(argA)`

Declaring an Interface	`interface IInterface`
	`{`
	`  void MyMethod();`
	`}`
		`interface IInterface`
		`  method MyMethod, void`
		`  endparams`
		`  endmethod`
		`endinterface`

Implementing an Interface	`class MyClass : IInterface`
	`{`
	`  public void MyMethod()`
	`  {`
	`  throw new Exception();`
	`  }`
	`}`
		`class MyClass implements IInterface`
		`  public method MyMethod, void`
		`  endparams`
		`  proc`
		`  throw new Exception()`
		`  endmethod`
		`endclass`

Auto-Implemented Property	`string MyString { get; set; }` `string MyString { get; }`	`readwrite property MyString, string` `readwrite property MyString, string`

Full Property	`private string _myString;`
	`public string MyString`
	`{`
	`  get`
	`  {`
	`  return _myString;`
	`  }`
	`  set`
	`  {`
	`  _myString = value;`
	`  }`
	`}`
		`private _myString, string`
		`public property MyString, string`
		`  method get`
		`  proc`
		`  mreturn _myString`
		`  endmethod`
		`  method set`
		`  proc`
		`  _myString = value`
		`  endmethod`
		`endproperty`

Structure	`struct MyStruct`
	`{`
	`  public string fname;`
	`  public string lname;`
	`  public string phoneNum;`
	`}`
		`cls structure MyStruct`
		`  public fname, string`
		`  public lname, string`
		`  public phoneNum, string`
		`endstructure`