StarCraft-GOAL Connector Manual

Harm Griffioen, Danny Plenge, Vincent Koeman

5^{th} June, 2018

Contents

Chapter 1

Connector

This connector is the first Environment Interface Standard (EIS) compatible connector that provides cognitive agents full access to StarCraft (Brood War). It brings the challenges of Real-Time Strategy (RTS) games to the field of multi-agent programming whilst also facilitating the development of AI solutions for such games, allowing the development of problem-solving techniques before being applied to similar but more complex real-world problems.

The design of this connector was guided by two conflicting objectives:

1. The connector should facilitate multi-agent systems that operate at a level of *abstraction* that is as high as possible.
2. The connector should facilitate multi-agent system implementations with as many different *strategies* as possible.

In other words, it does not aim for a multi-agent system that operates at the same level of detail as bots written in C or Java, but such a system should in contrast also not consist of a single action '*win*' that will delegate the control to some other subsystem instead. To make optimal use of the reasoning typically employed by cognitive agents, low-level details are handled in the environment whilst still allowing agents sufficiently fine grained control.

RTS games like StarCraft involve very large amounts of units that can come and go during the game and that have to deal with major challenges such as uncertainty and long-term (collaborative) goals, requiring multiple levels of abstraction and reasoning in the vast space of actions and game states that such games have. Therefore, a major factor that was also considered is the performance of the connector; a substantial performance impact caused by for example an enormous amount of percepts will limit the amount

of viable implementations (and thus possible strategies).

The remainder of this chapter will demonstrate how to set-up and start a bot with the StarCraft connector using a multi-agent system in the GOAL language. For the latest installation instructions, we refer to:
https://github.com/eishub/StarCraft/wiki/Install-Guide

1.1 Chaoslauncher

The Chaoslauncher facilitates plug-ins for StarCraft Brood War, like the *BWAPI Injector* which is necessary for using the BWAPI library that facilitates connecting to the internals of the game. It is also possible to make use of the *APMAlert* plugin, which shows the current actions per minute of all your units together. It is recommended to make use of the *W-Mode* plugin. This plugin automatically starts your StarCraft game in windowed mode which is easier for debugging. You can also make use of the *ChaosPlugin* to take advantage of its autoreplay function which automatically saves a replay at the end of each game. You can play these replays by turning off the *BWAPI Injector*, starting StarCraft (with the Chaoslauncher), selecting *Single Player* with gametype *Expansion*, pressing the 'Ok' button and then the 'Load Replay' button. If you then open the `Autoreplay` directory in that screen, you should be able to see all the replays which were saved by the autoreplay function. Alternatively, view replays in your browser at http://www.openbw.com/replay-viewer

1.2 Init Parameters

The StarCraft connector offers multiple configurable items through the init parameters of a mas2g file. When updating any parameters, do not forget to fully close the Chaoslauncher (i.e., closing it in the system tray) before launching a new game, as otherwise your changes will not be applied. The example below demonstrates all parameters and their defaults.

```
use "connector.jar" as environment with
    own_race="",
    enemy_race="random",
    map="",
    starcraft_location="C:\StarCraft",
    auto_menu="SINGLE_PLAYER",
    game_type="MELEE",
```

```
game_speed=50.
debug="false",
draw_mapinfo="false",
draw_unitinfo="false",
invulnernable="false",
managers=0,
percepts=[].
```

1.2.1 Own Race

You have to specify the race of your bot. This will make sure that the Chaoslauncher will automatically launch a game with the specified race. You can do this by inserting the following line: *own_race = <RaceName>*, where *<RaceName>* can either be *zerg*, *protoss*, *terran* or *random*. The option *random* will choose one race with a 1/3 chance for each race.

1.2.2 Enemy Race

The enemy race parameter can be used for specifying which race of the game's built-in AI you want to play against. To this end, you can insert *enemy_race=<RaceName>*, where *<RaceName>* can either be *zerg*, *protoss*, *terran*, *random*, *randomtp*, *randomtz*, or *randompz*. The option *random* will choose a race with a 1/3 chance for each race, whilst the other options will choose one of the two indicated races with a 1/2 chance for each race.

1.2.3 Map

You have to specify which map the Chaoslauncher will automatically load when starting the game. This can be done by inserting the following line: *map = <filename>*, where *<filename>* is the exact filename of the map (with extension). Please note that the environment only supports maps that are located in the directory *StarCraft/maps*, and that subdirectories (like *sscait*) should be indicated. Also note that the first time the environment runs on a certain map, it will take some time (around 2 minutes) to generate a datafile for the given map (if it is not already present in the *StarCraft/AI/BWTA* directory).

1.2.4 StarCraft Location

You have to specify the location of the StarCraft game if it is not installed in *C:/Starcraft*. Using this location, the Chaoslauncher will automatically start when launching a MAS. When the Chaoslauncher is already running, it will not start again until you close it (in the system tray), but this is fine as long as you use the same init parameters (although you have to start the next game manually in the existing Chaoslauncher instance then). You can specify the location of StarCraft by inserting *StarCraft_location = <FilePath>*, where *<FilePath>* is the absolute path to the StarCraft installation folder.

1.2.5 Auto Menu

The auto menu parameter is used to automatically go through the menus of the game when starting a MAS. This can be used for single player games and multi player games. To use the auto menu function you can insert the following line: *auto_menu=<MenuChoice>*, where *<MenuChoice>* can take the following values: *SINGLE_PLAYER*: for a single player game (against built-in AI), *LAN*: for a local multiplayer game, and *OFF*: no auto menu.

1.2.6 Game Type

The game type is used to indicate what kind of game the Chaoslauncher should start. Generally, you want this to be the default (*MELEE*), but other game types can be used by inserting *game_type=<GameType>*.

1.2.7 Game Speed

The game speed parameter can be used to set the initial speed of the game when the StarCraft game is launched (the speed can be changed during the game by using the development tool; see the next item). StarCraft makes use of a logical frame rate, which means that the game_speed depends on the amount of frames per second (fps) used to update the game. The higher the fps, the faster the game will go. For using the game_speed parameter you can insert the following line: *game_speed=<FPS>*, where *<FPS>*. If a number lower than 1 is given, there will be no limit on the amount of FPS used, and the game will run as fast as it possibly can on your CPU.

1.2.8 Debug

The environment offers a development tool for debugging purposes. With this development tool, you can increase or decrease the game speed, enable cheats and toggle the drawing of map and/or unit details in the game. More information about the development tool can be found in Section 1.4. In order to enable or disable launching the development tool, you can insert *debug=<Boolean>*.

1.2.9 Draw Map Info

This parameter can be used to draw info about the map (bases, regions, chokepoints) without having to enable it the development tool (or without starting the development tool at all) by inserting *draw_mapinfo=<Boolean>*.

1.2.10 Draw Unit Info

This parameter can be used to draw info about units (counts, IDs, health, targets) without having to enable it the development tool (or without starting the development tool at all) by inserting *draw_unitinfo=<Boolean>*.

1.2.11 Invulnerable

The invulnerable parameter can be used to automatically make your units invulnerable from the start of the game (which can also be done manually in the development tool). This can come in handy for testing purposes when you do not want to fight your opponent. To use the invulnerable function you can insert *invulnerable=<Boolean>*.

1.2.12 Managers

If higher than 0, the connector will generate the specified number of entities (N) with type 'manager' and name 'managerN' at the start of the game. Such manager entities will not receive any percepts by default, but can subscribe to global percepts (see the following subsection). Manager entities can even take a few actions: *cancel/1, debugdraw/1, debugdraw/3, forfeit/0, startManager/0*. To use managers you can thus insert *managers=<Integer>*.

1.2.13 Percepts

By default, the entity for any unit will only receive those percepts that are specific to it, i.e. generic unit and and unit-specific percepts (see the next chapter). Through this init parameter, however, it is possible to specify that units of a certain type (including managers) should receive certain global percepts (static or dynamic). This is possible through a list of lists, where the first element of each sublist is the entity type (see Section 4.2 and use the exact name for managers), and the following the names of the global percepts that should be received by corresponding units. For example, *percepts: [[manager1,base,chokepoint], [zergZergling,friendly,enemy]]* will ensure that the first manager will receive base and chokepoint percepts, whilst any zergling will receive friendly and enemy percepts.

1.3 Entity Types

When defining a launch rule it is important that a correct entity type is used (see Section 4.2). This value has to be the same type of the StarCraft unit without spaces and where the first letter is uncapitalised. So when you for example want to connect an agent to a `Terran SCV`, this can be done by using the entity type *terranSCV*. Note that each unit type starts with the race of the unit, followed by the exact name of the unit type, and please be aware that the environment will wait in the first game frame until *at least four actions* have been requested, e.g., until all initial workers have called *gather/1*. This will allow all initial agents (including managers) to fully start-up (and possible execute a few cycles already) before the game starts.

```
define myAgent as agent {
    ...
}
launchpolicy {
    when type = terranSCV launch myAgent.
}
```

With mind control (an advanced Protoss ability), units from other races can be taken over. These units will also get an entity. A possible way to accomodate such entities is by making sure any other unit type is connected to a generic agent through a wildcard launch rule at the end of your mas2g:

```
when type=* launch ...
```

1.4 The Development Tool

The development tool can be automatically launched by using the *debug* init parameter. It provides several actions that are useful for debugging purposes.

1.4.1 Game Speed

The Game Speed slider can be found at the top of the development tool window. When the slider is used, the speed of the game will be changed immediately. The slider always starts on a value of 50 fps (it will **not** reflect the *game_speed* init parameter if that was used). The slowest speed is 20 fps, and from there you can set it as fast as you want. Note that when the speed is set to more than 100 fps, your agents might act differently than they would on the tournament speed, as they have much less time to process each frame. Setting the game speed to more than 100 fps should thus only be used for quick testing purposes.

1.4.2 Cheat Actions

The development tool offers 6 buttons which instantly enable StarCraft cheats. Note that these cheats should be used for testing purposes only. The first cheat is called: *Give resources*, which gives the player 10000 minerals and 10000 gas. The second cheat is called: *Show map*, which makes the whole map visible for the player. Note that all your agents will then also perceive everything on the map. The third cheat is called: *Enemy attacks deal 0 damage*, which makes the units of the player immune for damage (note: this can be automatically enabled with the init parameter *invulnerable* as well). The fourth cheat is called: *Reduce build times*, which significantly reduces the times needed to construct a building or train a unit and even removes any time needed for performing upgrades. The fifth cheat is called: *No tech restrictions*, which removes any requirements on advancing in the tech tree, i.e., any unit or upgrade can always be constructed/trained/performed (if the required resources are there). The sixth and final cheat is called: *No supply cap*, which allows the player to exceed the maximum supply.

1.4.3 Draw Actions

The development tool can also be used to show map or unit details in StarCraft itself. There are 2 buttons to this end, reflecting the matching *draw_mapinfo* and *draw_unitinfo* init parameters. Please see the information above on these parameters for more information.

Chapter 2

Percepts

This chapter lists all the percepts that are generated by the StarCraft connector, which vary per unit and on the *percepts* init parameter (see the previous chapter). For the implementation of these percepts in your agent programs, we refer to the GOAL programming guide. Note that both buildings and moving soldiers are called 'units' in StarCraft.

In order to reduce the number of percepts, one generic guideline used in this connector is to *only create percepts for information that changes in a single match or between matches*. Even though there is a lot of static information in a game like StarCraft, like the type of a unit (i.e., biological or mechanic), what a certain unit costs to produce, or the units a certain building can produce, this information remains the same for any execution of any agent system, and is thus much better suited to be encoded in the agent system itself. To this end, a Prolog file is supplied in the connector's installer that contains a large list of predicates representing static information about the game. The predicates available in this file are listed at the end of this section. Note that it is not expected that agents 'hardcode' information about specific maps on which matches can be played, and thus agents will be informed about changes between matches (i.e., map-specific information).

Another guideline used in this connector is that *no data is sent through percepts that can either be calculated based on other data, (e.g., the number of friendly units by adding the amount of percepts about their status) or retrieved from other agents (e.g., the position of a friendly unit)*. Relaying such information through messaging(channels) is usually much more efficient, as one can then selectively choose at which times and to which units to send information, as opposed to percepts always being sent to certain units even when they do not require them (at that time) for their decision making.

2.1 Global Static Percepts

These percepts represent global information (i.e., not specific to a certain unit) that will not change during a match. Note that all coordinates (X,Y) reflect tile positions; one such tile is actually 16 by 16 pixels.

2.1.1 base/6

Description	Information about all base locations on the map. These are possible construction sites for resource centers (or spawning sites for the initial centers).
Type	Send once
Syntax	`base(<IsPossibleStart>,<Minerals>,<Gas>,<X>,<Y>,<Region>)`
Example	`base(true, 13500, 5000, 28, 32, 8)`

Parameters

`<IsPossibleStart>`	Indicates whether the location is a potential starting location or not (i.e., there are 2 on a 2-player map and 4 on a 4-player map).
Type	`Boolean`
Range	[true,false]
`<Minerals>`	The total amount of minerals available at/near the base location (at the start of the game).
Type	`Integer`
Range	[0–∞]
`<Gas>`	The total amount of vespense gas available at/near the base location (at the start of the game).
Type	`Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the base location.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the base location.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the base is located in, which can for example be used to find matching resources.
Type	`Integer`
Range	[1–∞]

2.1.2 chokepoint/6

Description: Information about all chokepoints on the map. These are the narrow points on the map where only a limited amount of units can go through at the same time (depending on the specific chokepoint's width). All regions on the map are connected through chokepoints.

Type: Send once

Syntax: `chokepoint(<X1>,<Y1>,<X2>,<Y2>,<Region1>,<Region2>)`

Example: `chokepoint(12, 15, 14, 17, 1, 2)`

Parameters:

`<X1>`	The x-coordinate of the first side.
Type	`Integer`
Range	[0–∞]
`<Y1>`	The y-coordinate of the first side.
Type	`Integer`
Range	[0–∞]
`<X2>`	The x-coordinate of the second side.
Type	`Integer`
Range	[0–∞]
`<Y2>`	The y-coordinate of the second side.
Type	`Integer`
Range	[0–∞]
`<Region1>`	The ID of the first region.
Type	`Integer`
Range	[1–∞]
`<Region2>`	The ID of the second region.
Type	`Integer`
Range	[1–∞]

2.1.3 enemyPlayer/2

Description	The name and race of the opponent (it is assumed there is only one opponent in this connector).
Type	Send once
Syntax	`enemyPlayer(<Name>,<Race>)`
Example	`enemyPlayer('ForceBot', zerg)`
Note	When playing against a random race (*<Race>=unknown*), you can use something like:

```
if bel(enemyPlayer(Name,unknown), enemy(_,Type,_,_,_,_,_,_,_,_,_,_),
sub_string(Type,0,1,_,Race)) then {
  if bel(Race = "z") then delete(enemyPlayer(Name,unknown))
  + insert(enemyPlayer(Name,zerg)).
  if bel(Race = "p") then delete(enemyPlayer(Name,unknown))
  + insert(enemyPlayer(Name,protoss)).
  if bel(Race = "t") then delete(enemyPlayer(Name,unknown))
  + insert(enemyPlayer(Name,terran)).
 }
```

Parameters		
	<Name>	The name of the enemy player.
	Type	`String`
Parameters	**<Race>**	The enemy race.
	Type	`String`
	Range	*[terran,protoss,zerg,unknown]*

2.1.4 map/3

Description	The name, width, and the height of the map (in tiles).
Type	Send once
Syntax	`map(<Name>,<Width>,<Height>)`
Example	`map('Destination 1.1', 96, 128)`

Parameters		
	<Name>	The name of the map (ASCII characters only).
	Type	`String`
	<Width>	The width of the map (no. of horizontal tiles).
	Type	`Integer`
	Range	[1–∞]
	<Height>	The height of the map (no. of vertical tiles).
	Type	`Integer`
	Range	[1–∞]

2.1.5 ownRace/1

Description	The race of yourself (useful when playing as random).
Type	Send once
Syntax	`ownRace(<Race>)`
Example	`ownRace(protoss)`

Parameters

`<Race>`	The player's race.
Type	`String`
Range	*[terran,protoss,zerg]*

2.1.6 region/5

Description	Information about all regions on the map. Regions are connected by chokepoints and can be on high or low ground.
Type	Send once
Syntax	`region(<Id>,<CenterX>,<CenterY>,<Height>,<ConnectedRegionsList>)`
Example	`region(12, 15, 14, 17, [1,2])`

Parameters

`<Id>`	The ID of the region.
Type	`Integer`
Range	[1–∞]
`<CenterX>`	The x-coordinate of the center of the region.
Type	`Integer`
Range	[0–∞]
`<CenterY>`	The y-coordinate of the center of the region.
Type	`Integer`
Range	[0–∞]
`<Height>`	The (relative) height of the region.
Type	`Integer`
Range	[0–∞]
`<ConnectedRegionsList>`	A list of regions (by ID) that are connected to this region (i.e., through chokepoints).
Type	`List`

2.2 Global Dynamic Percepts

These percepts represent information that changes during a match, but is still global to the match (i.e., not specific to a certain unit). Note that units will keep the same ID after a morph or merge.

2.2.1 attacking/2

Description	All enemy units that are attacking / going to attack and the corresponding friendly units that they have targeted.
Type	Send always
Syntax	`attacking(<Id>,<TargetId>)`
Example	`attacking(123, 177)`

Parameters

`<Id>`	The ID of the enemy unit that is attacking / going to attack.
Type	`Integer`
Range	[1–∞]
`<TargetId>`	The ID of the friendly unit that is being targeted.
Type	`Integer`
Range	[1–∞]

2.2.2 constructionSite/3-4

Description: All visible and non-obstructed locations at which buildings can potentially be constructed. Such construction sites are squares containing 4 tiles, as the minimum size of any building is 2 by 2 tiles. This information is updated every 50 game frames only. Note that resource centers require a minimum distance to mineral patches and geysers (which the locations as indicated by *base/4* conform to for example).

Type: Send always

Syntax: (Protoss) `constructionSite(<X>,<Y>,<Region>,<InPylonRange>)`
(Zerg) `constructionSite(<X>,<Y>,<Region>,<OnCreep>)`
(Terran) `constructionSite(<X>,<Y>,<Region>)`

Example:
```
constructionSite(66, 98, 4, false)
constructionSite(66, 98, 4)
```

Parameters:

<X>	The x-coordinate of the construction site.
Type	`Integer`
Range	[0–∞]
<Y>	The y-coordinate of the construction site.
Type	`Integer`
Range	[0–∞]
<Region>	The region the construction site is in.
Type	`Integer`
Range	[1–∞]
<InPylonRange>	Indicates whether the construction site is in range of a pylon (for Protoss only).
Type	`Boolean`
Range	[true,false]
<OnCreep>	Indicates whether the construction site is on creep (for Zerg only).
Type	`Boolean`
Range	[true,false]

2.2.3 enemy/11

Description: Information about all enemy units. Note that this also includes unfinished units (like buildings under construction), cloaked units that cannot be attacked until they are detected (see *<Conditions>*), or even units that cannot be attacked at all (like spells). Moreover, this percept will be sent for any enemy unit that has ever been visible (see *<LastUpdated>*); percepts will no longer be sent for only those enemy units that were destroyed in our view. Neutral buildings on the map are also regarded as enemy units.

Type: Send always

Syntax:
```
enemy(<Id>,<Type>,<Health>,<Shield>,<Energy>,<Conditions>
  <Orientation>,<X>,<Y>,<Region>,<LastUpdated>)
```

Example:
```
enemy(12, 'Zerg Overlord', 100, 0, 0, [flying], 0, 12, 96, 3, 50)
```

Parameters:

<Id>	The ID of the unit.
Type	Integer
Range	[1–∞]
<Type>	The type of the unit. This consists of a string with the race of the unit and the name of the unit parted by a space.
Type	String
Range	See 4.2
<Health>	The current amount of health of the unit.
Type	Integer
Range	[1–<maxHealth>]
<Shield>	The current amount of shields of the unit.
Type	Integer
Range	[0–<maxShield>]
<Energy>	The current amount of energy of the unit.
Type	Integer
Range	[0–<maxEnergy>]
<Conditions>	A list representing the current conditions of the unit. Each unit can have multiple or no conditions depending on the unit and situation. Note that not all conditions are available for enemies.
Type	List of Strings
Range	See 2.5
<Orientation>	The orientation of the unit, rounded to the nearest multiple of 45 degrees.
Type	Integer
Range	[0–360]
<X>	The x-coordinate of the unit.
Type	Integer
Range	[0–∞]
<Y>	The y-coordinate of the unit.
Type	Integer
Range	[0–∞]

Continued

<Region>	The region the unit is in. Can be 0 if the unit is on a chokepoint (and thus 'in-between' regions).
Type	`Integer`
Range	[0–∞]
<LastUpdated>	The game frame in which information about this unit was last updated. If this is equal to the current game frame, the enemy unit is within our view; otherwise it is within the fog-of-war. Only enemy units that are destroyed within our view are removed from the percepted enemies.
Type	`Integer`
Range	[0–∞]

2.2.4 friendly/2

Description: Information about all (living) units of the player. Note that this also includes unfinished units that do not have an entity and thus agent yet (like buildings under construction), see also the *underConstruction/6* percept.

Type: Send always

Syntax: `friendly(<Id>,<Type>)`

Example: `friendly(26, 'Protoss Gateway')`

Parameters

<Id>	The ID of the unit.
Type	`Integer`
Range	[1–∞]
<Type>	The type of the unit. This consists of a string with the race of the unit and the name of the unit parted by a space.
Type	`String`
Range	See 4.2

2.2.5 gameframe/1

Description: The current game frame. For more information see 1.2.7.

Type: Send on change

Syntax: `gameframe(<Number>)`

Example: `gameframe(150)`

Parameters

<Number>	The game frame count.
Type	`Integer`
Range	[0–∞]

2.2.6 mineralField/5

Description	Information about visible (non-empty) mineral fields.
Type	Send always
Syntax	`mineralField(<Id>,<Resources>,<X>,<Y>,<Region>)`
Example	`mineralField(57, 5000, 6, 22, 32)`

Parameters

`<Id>`	The ID of the mineral field.
Type	`Integer`
Range	[1–∞]
`<Resources>`	The amount of minerals left in the field, rounded to the nearest (upper) multiple of 100. Can be 0 for mineral fields that obstruct e.g. chokepoints (mined out fields disappear).
Type	`Integer`
Range	[0–5000]
`<X>`	The x-coordinate of the mineral field.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the mineral field.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the mineral field is in.
Type	`Integer`
Range	[1–∞]

2.2.7 nuke/3

Description	Indicates that a nuclear strike will land on the given position.
Type	Send always
Syntax	`nuke(<X>,<Y>,<Region>)`
Example	`nuke(22, 37, 1)`

Parameters

<X>	The x-coordinate of the incoming nuclear strike.
Type	`Integer`
Range	[0–∞]
<Y>	The y-coordinate of the incoming nuclear strike.
Type	`Integer`
Range	[0–∞]
<Region>	The corresponding region for the incoming nuclear strike.
Type	`Integer`
Range	[1–∞]

2.2.8 researched/1

Description	A list of all upgrade types that have been completed.
Type	Send on change
Syntax	`researched(<CompletedList>)`
Example	`researched(['Grooved Spines', 'Lurker Aspect'])`

Parameters

<CompletedList>	All finished upgrades.
Type	`List`
Range	See Section 4.3

2.2.9 resources/4

Description: The amount of minerals, gas and supply available to the player (i.e. shared by all units). Note that in order to avoid halves, supply is multiplied by 2 throughout this interface, so 10 supply in-game corresponds with 20 supply in this environment.

Type: Send on change

Syntax: `resources(<Minerals>,<Gas>,<UsedSupply>,<TotalSupply>)`

Example: `resources(350, 100, 25, 41)`

Parameters:

`<Minerals>`	The current amount of minerals available.
Type	`Integer`
Range	[0–∞]
`<Gas>`	The current amount of gas available.
Type	`Integer`
Range	[0–∞]
`<UsedSupply>`	The amount of supply that is in use
Type	`Integer`
Range	[0–400]
`<TotalSupply>`	The total amount of supply that is available. Note that usually `<TS>` is always greater or equal to `<CS>`, but this can change when supply providing units are killed.
Type	`Integer`
Range	[0–400]

2.2.10 underConstruction/6

Description: Indicates a new friendly unit (i.e. either a building or a moving unit) that is under construction (i.e. by another unit, or morphing, or warping in). This percept allows getting information about unfinished units that do not have an entity (and thus agent) yet.

Type: Send always

Syntax: `underConstruction(<Id>,<BuilderId>,<Vitality>,<X>,<Y>,<Region>)`

Example: `underConstruction(44, 3, 74, 22, 37, 2)`

Parameters:

`<Id>`	The ID of the (unfinished) unit.
Type	`Integer`
Range	[1–∞]
`<Id>`	The ID of the unit that is responsible for the construction or training of this unit (if any applicable; -1 otherwise).
Type	`Integer`
Range	[-1–∞]
`<Vitality>`	The combined amount of health and shield of the unit. This generally increases whilst the unit is nearing completion, though it can simultaneously be brought down as well by enemy attacks.
Type	`Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the unit (cannot change).
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit (cannot change).
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the unit is in. Can be 0 if for example a morphing unit is on a chokepoint (and thus 'in-between' regions).
Type	`Integer`
Range	[0–∞]

2.2.11 vespeneGeyser/5

Description	Information about visible (though possibly empty) vespene geysers. Empty geysers can still be mined from, though at a reduced rate.
Type	Send always
Syntax	`vespeneGeyser(<Id>,<Resources>,<X>,<Y>,<Region>)`
Example	`vespeneGeyser(57, 5000, 22, 32, 6)`

Parameters

`<Id>`	The ID of the vespene geyser.
Type	`Integer`
Range	[1–∞]
`<Resources>`	The amount of gas left in the vespene geyser, rounded to the nearest (upper) multiple of 100. Can be 0, after which mining workers will obtain much less gas per trip.
Type	`Integer`
Range	[0–∞]
`<X>`	The x-coordinate of the vespene geyser.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the vespene geyser.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the vespene geyser is in.
Type	`Integer`
Range	[1–∞]

2.2.12 winner/1

Description	Indicates if the player has won or lost at the end of the game. Used mainly for automated testing purposes.
Type	Send once
Syntax	`winner(<HasWon>)`
Example	`winner(true)`

Parameters

`<HasWon>`	Whether the player has won or not.
Type	`Boolean`
Range	[true,false]

2.3 Generic Unit Percepts

These percepts are generated for all individual units in the game (thus excluding managers). Although the *self* percept represents static information that does not change during the match, the *status* and the *order* percepts are updated frequently.

2.3.1 self/2

Description	Indicates the ID and type of the unit itself.
Type	Send once
Syntax	`self(<Id>,<Type>)`
Example	`self(21, 'Terran SCV')`

Parameters

`<Id>`	The ID of the unit.
Type	`Integer`
Range	[1–∞]
`<Type>`	The type of the unit. This consists of a string with the race of the unit and the name of the unit parted by a space.
Type	`String`
Range	See Section 4.2

2.3.2 status/8

Description The current amount of health, shield and energy of the unit. The `status` percept also shows the conditions of the unit and its current orientation and position.

Type Send on change

Syntax `status(<Health>,<Shield>,<Energy>,<Conditions>, <Orientation>,<X>,<Y>,<Region>)`

Example `status(250, 0, 0, [moving, carrying], 180, 24, 36, 1)`

Parameters

`<Health>`	The current amount of health of the unit.
Type	`Integer`
Range	[0–`<MaxHealth>`] where `<MaxHealth>` is the maximum health of the given unit.
`<Shield>`	The current amount of shields of the unit.
Type	`Integer`
Range	[0–`<MaxShield>`] where `<MaxShield>` is the maximum shield of the given unit.
`<Energy>`	The current amount of energy of the unit.
Type	`Integer`
Range	[0–`<MaxEnergy>`] where `<MaxEnergy>` is the maximum energy of the given unit.
`<Conditions>`	A list representing the current conditions of the unit. Each unit can have multiple or no conditions depending on the unit and situation.
Type	`List of Strings`
Range	See Section 2.5
`<Orientation>`	The orientation of the unit, rounded to the nearest multiple of 45 degrees.
Type	`Integer`
Range	[0–360]
`<X>`	The x-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Y>`	The y-coordinate of the unit.
Type	`Integer`
Range	[0–∞]
`<Region>`	The region the unit is in. Can be 0 if the unit is on a chokepoint (and thus 'in-between' regions).
Type	`Integer`
Range	[0–∞]

2.3.3 order/6

Description: Indicates what order(s) a unit is executing. A unit always has an order (e.g., even 'Nothing' is an order).

Type: Send on change

Syntax: `order(<Primary>,<TargetUnit>,<TargetX>,<TargetY>,<TargetRegion>,<Secondar`

Example: `order('AttackMove', -1, 34, 8, 5, 'None')`

Parameters:

<Primary>	The primary order of the unit. Some actions are converted into race or unit specific orders.
Type	`String`
Range	See https://bwapi.github.io/namespace_b_w_a_p_i_1_1_orders.html
<TargetUnit>	The ID of the unit the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
<TargetX>	The X coordinate of the position the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
<TargetY>	The Y coordinate of the position the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
<TargetRegion>	The region of the position the order is targeted at if any; -1 otherwise.
Type	`Integer`
Range	[-1–∞]
<Secondary>	The secondary order of the unit. This is usually 'None', but is used when for example a Protoss Carrier is both moving and producing units.
Type	`String`
Range	See https://bwapi.github.io/namespace_b_w_a_p_i_1_1_orders.html

2.4 Unit-Specific Percepts

These percepts are generated only for specific units in the game. All of these (dynamic) percepts represent information that can change during the match.

2.4.1 defensiveMatrix/1

Description	Information about how much health the defensive matrix has left on the unit. This only applies to friendly Terran units having received such a matrix from a Science Vessel.
Type	Send always
Syntax	`defensiveMatrix(<Health>)`
Example	`defensiveMatrix(200)`

Parameters

`<Health>`	The amount of health left for the defensive matrix.
Type	`Integer`
Range	[0–250]

2.4.2 queueSize/1

Description	The number of units that are in the queue of any production unit, thus including e.g. Protoss Carriers. (Zerg Hatchery/Lair/Hive) the number of of available larva. (Terran Nuclear Silo) 1 if a nuke is ready (after a corresponding *train/1* action); 0 otherwise. (Terran Vulture) the number of spider mines left (which are automatically granted upon production of the vulture). (Protoss Carrier) the number of interceptors (ready and in production). (Protoss Reaver) the number of scarabs (ready and in production).
Type	Send on change
Syntax	`queueSize(<Size>)`
Example	`queueSize(2)`

Parameters

`<Size>`	See description.
Type	`Integer`
Range	[0–5]

2.4.3 researching/1

Description	Indicates which upgrade is being researched by the unit (i.e., a building).
Type	Send always
Syntax	`researching(<Type>)`
Example	`researching('Stim Packs')`

Parameters

<Type>	The upgrade that is being researched.
Type	`String`
Range	See Section 4.3

2.4.4 unitLoaded/1

Description	Indicates which unit(s) are loaded inside the loadable unit (e.g. a Terran Bunker or a Protoss Shuttle).
Type	Send always
Syntax	`unitLoaded(<Id>)`
Example	`unitLoaded(154)`

Parameters

<Id>	The ID of the loaded unit.
Type	`Integer`
Range	[0–∞]

2.5 Conditions

This section lists the conditions a unit can potentially have. The race-specific conditions are either only applicable to or caused by units from that race.

2.5.1 Workers

`carrying`	Indicates when a worker unit is carrying minerals or vespene gas.
`constructing`	Shows that a worker unit is busy constructing a building.
`gathering`	Show that a worker unit is busy gathering minerals or vespene gas.
`repairing`	Shows that a (Terran) worker unit is busy repairing a building.

2.5.2 Generic

`attacking`	Indicates when a unit is attacking an other unit (includes medic heal).
`beingConstructed`	Indicates that a unit is incomplete (includes morphing).
`cloaked`	Indicates that a unit is cloaked.
`coolingDown`	Indicates that a unit cannot attack due to cooldown.
`detected`	Indicates that an enemy cloaked/burrowed unit has been detected (and thus can be attacked).
`flying`	Indicates that a unit is flying.
`following`	Indicates that a unit is following an other unit.
`holding`	Indicates that a unit is holding a position.
`idle`	Indicates that the unit is idle (not doing anything).
`loaded`	Indicates that a unit is loaded (i.e. has one or more units in it).
`moving`	Indicates that a unit is moving.
`patrolling`	Indicates that a unit is patrolling between 2 positions.
`underAttack`	Indicates that a unit is under attack.

2.5.3 Zerg

`acidSpored`	Indicates that a unit is under Acid Spores from a Defiler.
`burrowed`	Indicates that a unit is burrowed.
`darkSwarmed`	Indicates that a unit is under a Dark Swarm from a Defiler.
`ensnared`	Indicates that a unit is ensnared by a Queen.
`morphing`	Indicates that a unit is morphing.
`parasited`	Indicates that a unit is parasited by a Queen.
`plagued`	Indicates that a unit is plagued by a Defiler.

2.5.4 Terran

`<addonName>`	Indicates that an addon of a building is present.
`beingHealed`	Indicates that a unit is being healed by a Medic or repaired by a SCV.
`blinded`	Indicates that a unit is blinded by a Medic.
`defenseMatrixed`	Indicates that a unit has a defensive matrix on it (from a Science Vessel).
`hasMines`	Indicates that a Vulture has at least one Spider Mine.
`irradiated`	Indicates that a unit is irradiated by a Science Vessel.
`lifted`	Indicates that a building is lifted (and thus can move).
`lockDowned`	Indicates that a unit is under lockdown by Ghost.
`nukeReady`	Indicates that a Nuclear Silo is ready to launch a Nuclear Missile.
`sieged`	Indicates that a Siegetank is in siegemode.
`stimmed`	Indicates that a Firebat or Marine is stimmed.

2.5.5 Protoss

`disruptionWebbed`	Indicates that a unit is in a disruption web from a Corsair.
`hallucination`	Indicates that a friendly unit is a hallucination (from a High Templar)
`hasInterceptors`	Indicates that a Carrier has at least one Interceptor.
`hasScarabs`	Indicates that a Reaver has at least one Scarab.
`maelstrommed`	Indicates that a unit is maelstrommed by a Dark Archon.
`stasised`	Indicates that a unit is stuck in stasis from an Arbiter.
`underStorm`	Indicates that a unit is under a storm from a High Templar.
`unpowered`	Indicates that a building is no longer powered by a pylon (e.g. a Photon Cannon then no longer functions).

Chapter 3

Actions

This chapter lists all the actions that are usable in the StarCraft environment, which vary per unit.

3.1 All Units

The following actions can be executed by any agent (i.e., including managers).

3.1.1 cancel/1

Description	Cancel the construction or morphing of a unit.
Syntax	`cancel(<TargetId>)`
Example	`cancel(3)`
Parameters	`<TargetId>`: The ID of the unit of which the construction or morphing should be cancelled.
Pre	The targeted unit is incomplete (not fully constructed or morphed).
Post	The targeted unit's construction or morphing will be cancelled; 75% of the invested resources will be refunded (and for Zerg the original unit will be restored).

3.1.2 debugdraw/1

Description	Draw text above the unit in the game window.
Syntax	`debugdraw(<Text>)`
Example	`debugdraw("Power Overwhelming")`
Parameters	`<Text>`: The text(string) that should be drawn; this can include characters like tabs and newlines.
Pre	-
Post	The given text will be drawn above the unit (i.e., it will stay with the unit in the game window). If the given text is empty, any existing drawing will be cancelled.
Note	For managers, the text will be drawn on a fixed position on the left top of the game window.

3.1.3 debugdraw/2

Description	Draw text above any unit in the game window.
Syntax	`debugdraw(<TargetId>,<Text>)`
Example	`debugdraw(5, "Power Overwhelming")`
Parameters	`<TargetId>`: The ID of the (possibly enemy or even neutral) unit above which the text should be drawn. `<Text>`: The text(string) that should be drawn; this can include characters like tabs and newlines.
Pre	-
Post	The given text will be drawn above the unit (i.e., it will stay with the unit in the game window), as long as it is visible (or becomes visible again). If the given text is empty, any existing drawing will be cancelled.

3.1.4 debugdraw/3

Description	Draw text on a specific location on the map (in the game window).
Syntax	`debugdraw(<X>,<Y>,<Text>)`
Example	`debugdraw(1, 23, "Important Location")`
Parameters	`<X>`: The X coordinate of the location to draw on. `<Y>`: The Y coordinate of the location to draw on. `<Text>`: The text(string) that should be drawn; this can include characters like tabs and newlines.
Pre	-
Post	The given text will be drawn on the given location (i.e., it will stay on the location in the game window). If the given text is empty, any existing drawing will be cancelled.

3.1.5 forfeit/0

Description	Forfeit the game.
Syntax	`forfeit`
Pre	The game is in progress.
Post	The game ends with a loss for the player.

3.1.6 morph/1 (Zerg units only)

Description	Morph a unit into another unit(type).
Syntax	`morph(<Type>)`
Example	`morph('Zerg Lurker')`
Parameters	`<Type>`: The type to morph into. See Section 4.2.
Pre	The unit is capable of morphing into the given unit type.
Post	The unit's corresponding agent terminates and a new agent is created for the new unit when it is completed (with the same ID).

3.1.7 startNewManager/0

Description	Starts a new manager entity.
Syntax	`startNewManager`
Pre	-
Post	A new manager entity is launched (with a number that follows the existing sequence of mangers). To stop a manager entity, the corresponding agent's main module should be terminated (which can be done with either an exit condition or an exit-module action).

3.2 Buildings

The actions in this section can only be executed by buildings (or by some special units that can be loaded or that can produce units of their own).

3.2.1 buildAddon/1 (Terran only)

Description	Build an addon.
Syntax	`buildAddon(<Name>)`
Example	`buildAddon('Terran Comsat Station')`
Parameters	`<Name>`: The name of the addon that is to be constructed. See Section 4.2.
Pre	The building is capable of building the addon and does not already have an addon.
Post	The building starts constructing the addon.

3.2.2 cancel/0

Description	Cancel the last train or research action.
Syntax	`cancel`
Pre	The unit is training, researching, or constructing an add-on (Terran-only).
Post	The last train, research, or add-on build is cancelled; the resources are refunded.

3.2.3 land/2 (Terran only)

Description	Land a lifted building on a given location.
Syntax	`land(<X>,<Y>)`
Example	`land(22, 33)`
Parameters	`<X>`: The x-coordinate of the chosen landing location. `<Y>`: The y-coordinate of the chosen landing location.
Pre	The unit is currently lifted and the landing location is visible, not obstructed, and fitting for the building.
Post	The unit moves to (if needed) and lands on the chosen location. It reconnects with any addon if applicable.

3.2.4 lift/0 (Terran only)

Description	Lift a building into the air.
Syntax	`lift`
Pre	The unit is capable of lifting and is not currently performing any other action.
Post	The building lifts into the air.

3.2.5 load/1

Description	Load a given unit into the unit.
Syntax	`load(<Id>)`
Example	`load(2)`
Parameters	`<Id>`: The ID of the unit to load into this unit.
Pre	The unit is capable of loading the targeted unit and has enough space provided for the targeted unit.
Post	The targeted unit moves towards to the loadable unit and loads into it.

3.2.6 research/1

Description	Research an upgrade.
Syntax	`research(<Type>)`
Example	`research('Cloaking Field')`
Parameters	`<Type>`: The type of upgrade to research. See Section 4.3.
Pre	The unit is capable of researching the given upgrade.
Post	The unit starts researching the given upgrade.
Note	The level of an upgrade (if applicable) is optional; this stacks automatically.

3.2.7 train/1

Description	Train a unit.
Syntax	`train(<Type>)`
Example	`train('Protoss Zealot')`
Parameters	`<Type>`: The type of unit to train. See Section 4.2.
Pre	The unit is capable of producing the given unit.
Post	The unit starts producing the given unit.

3.2.8 unload/1

Description	Unload a loaded unit from the unit.
Syntax	`unload(<Id>)`
Example	`unload(3)`
Parameters	`<Id>`: The ID of the unit to unload from this unit.
Pre	The given unit is currently loaded into the unit.
Post	The targeted unit is unloaded from the unit.

3.2.9 unloadAll/0

Description	Unload all loaded units from the unit.
Syntax	`unloadAll`
Pre	There are units currently loaded into the unit.
Post	All loaded units are unloaded from the unit.

3.3 Moving Units

The action in this section can only be executed by moving units (i.e. non-buildings or lifted Terran buildings).

3.3.1 ability/1

Description	Use an ability.
Syntax	`ability(<Type>)`
Example	`ability('Burrowing')`
Parameters	`<Type>`: The type of ability to use. See Section 4.4.
Pre	The unit is capable of performing the ability (on itself, i.e., without a target unit or location).
Post	The unit performs the ability.
Note	Behaviour that can be toggled on and off (e.g. Burrow/-Cloak/Siege) is also executed by using this action (i.e. once for enabling and then again for disabling).

3.3.2 ability/2

Description	Use an ability on a target unit.
Syntax	`ability(<Type>,<Target>)`
Example	`ability('Archon Warp', 3)`
Parameters	`<Type>`: The type of ability to use. See Section 4.4. `<Target>`: The target to use the technology on.
Pre	The unit is capable of performing the ability (with some target unit), and the target unit is visible.
Post	The unit performs the ability on the target unit.

3.3.3 ability/3

Description	Use an ability on a location.
Syntax	`ability(<Type>, <X>, <Y>)`
Example	`ability('Dark Swarm', 11, 8)`
Parameters	`<Type>`: The type of ability to use. See Section 4.4. `<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	The unit is capable of performing the ability (with some target location).
Post	The unit performs the ability (on the given location), first moving closer to the location if required.

3.3.4 attack/1

Description	Attack a given unit.
Syntax	`attack(<TargetId>)`
Example	`attack(12)`
Parameters	`<TargetId>`: The ID of the unit that should be attacked.
Pre	The unit is attack capable and the targeted unit is visible and reachable.
Post	The targeted unit is being attacked by your unit. The unit will keep moving towards the enemy unit in order to attack it as long as it is visible and alive.
Note	Terran Medics can use this action to heal friendly units; they cannot attack enemies.

3.3.5 attack/2

Description	Move to a given location and attack everything on the way.
Syntax	`attack(<X>,<Y>)`
Example	`attack(9, 21)`
Parameters	`<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	The unit is attack capable.
Post	The unit moves to the chosen location (or as close as it can get) whilst attacking any enemy unit that it encounters along the way; all such enemy units will be chased until they are no longer visible or alive. Neutral units (that are percepted as enemy) will not be automatically attacked with this action.
Note	Terran Medics will heal any friendly units they encounter.

3.3.6 follow/1

Description	Follow a given unit.
Syntax	`follow(<given>)`
Example	`follow(5)`
Parameters	`<given>`: The ID of the unit that should be followed.
Pre	The targeted unit is visible.
Post	The unit follows the selected unit; any enemy will be ignored (i.e. the unit will not automatically attack anything).

3.3.7 hold/0

Description	Hold a position.
Syntax	`hold`
Pre	-
Post	The unit will hold its current position; any enemy will be ignored (i.e. the unit will not automatically attack anything).

3.3.8 move/2

Description	Move to a given location.
Syntax	`move(<X>,<Y>)`
Example	`move(19, 1)`
Parameters	`<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	-
Post	The unit moves to the chosen location (or as close as it can get) whilst ignoring any enemy unit along the way (i.e. the unit will not automatically attack anything).

3.3.9 patrol/2

Description	Patrol between a unit's current position and the given location.
Syntax	`patrol(<X>,<Y>)`
Example	`patrol(7,8)`
Parameters	`<X>`: The x-coordinate of the chosen location. `<Y>`: The y-coordinate of the chosen location.
Pre	-
Post	The unit patrols between its current position and the chosen location (or as close as it can get); any enemy unit that it encounters will be chased until it is no longer visible or alive, after which the unit will return to its patrol route.
Note	Terran Medics will heal any friendly units they encounter.

3.3.10 stop/0

Description	Stop performing an action.
Syntax	`stop`
Pre	The unit is performing some kind of action.
Post	The unit stops performing its current action.

3.4 Workers

The actions in this section can only be executed by worker units.

3.4.1 build/3

Description	Build a building on the given location.
Syntax	`build(<Type>,<X>,<Y>)`
Example	`build('Terran Supply Depot', 24, 6)`
Parameters	`<Type>`: The type of building that should be built. See 4.2. `<X>`: The x-coordinate of the build location. `<Y>`: The y-coordinate of the build location.
Pre	The unit is capable of constructing the chosen building.
Post	The unit goes moves the build location (if needed) and starts constructing the building there if this is possible considering the location and the requested building. Zerg Drones will morph (i.e., the drone will be lost), Terran SCVs will be busy constructing for a while, and Protoss Probes will instantiate a warp (i.e., the probe does not have to remain at the build location). See also *cancel/1* and *repair/1*.
Note	The eventual location of the building may not match the location given in the action, as constructions are started at the left top of a grid, but building locations are given by their center (and thus depend on their size).

3.4.2 gather/1

Description	Gather a specific resource (i.e., from a mineral cluster or a vespene gas building).
Syntax	`gather(<Id>)`
Example	`gather(32)`
Parameters	`<Id>`: The ID of the chosen resource.
Pre	-
Post	The unit starts gathering the chosen resource (if it can reach it). It automatically moves back and forth between the resource and the closest resource center.

3.4.3 repair/1 (Terran only)

Description	Repair a unit or complete an unfinished building.
Syntax	`repair(<Id>)`
Example	`repair(17)`
Parameters	`<Id>`: The ID of the unit to repair or of the building to complete construction of.
Pre	The unit is a Terran SCV, has the resources to repair, and the target unit is visible (and reachable).
Post	The SCV moves towards the selected unit (if needed) and repairs it or resumes its construction.

Chapter 4

Knowledge

This chapter lists the knowledge predicates that are supplied in a Prolog file with all the example agents in the connector's installer. These represent static knowledge about StarCraft that does not change in between matches. For easy reference, this section also includes a list of all unit and upgrade types per race (and where they can be trained/researched). Moreover, the upgrade types that have a corresponding ability are listed at the end (including by which unit they can be used and at what target). We end with listing some of the known limitations of the connector.

4.1 Predicates

4.1.1 unit/2

Description: Indicates all possible unit types. If a matching *combat/6* predicate exists, units of this type are attack capable.

Syntax: `unit(<Name>,<Race>)`

Example: `unit('Protoss Reaver',protoss)`

Parameters:

`<Name>`	The full name of the unit type (i.e. as used in actions and percepts).
Type	`String`
Range	See Section 4.2
`<Race>`	The race the unit type belongs to.
Type	`String`
Range	[terran,zerg,protoss]

4.1.2 upgrade/2

Description: Indicates all possible upgrade types. If a matching *combat/6* predicate exists, the upgrade represents an ability. Not all abilities actually need their matching upgrade to be researched at a building first (costs in *costs/6* will be zero for such abilities/upgrades).

Syntax: `upgrade(<Name>,<Race>)`

Example: `upgrade('Tank Siege Mode',terran)`

Parameters:

`<Name>`	The full name of the upgrade type (i.e. as used in the research actions and percepts).
Type	`String`
Range	See Section 4.3
`<Race>`	The race the upgrade type belongs to.
Type	`String`
Range	[terran,zerg,protoss]

4.1.3 costs/6

Description: Information about the costs in resources, time and existing units or upgrades of a unit or upgrade type.

Syntax:
```
costs(<Name>,<Minerals>,<Gas>,<SupplyOrEnergy>,
  <BuildFrames>,<RequiredUnitsOrUpgrades>)
```

Example:
```
costs('Zerg Lurker', 50, 100, 4, 600,
  ['Lurker Aspect','Zerg Hydralisk']).
```

Parameters

<Name>	The name of the unit or upgrade type.
Type	`String`
Range	See Sections 4.2 and 4.3
<Minerals>	The required amount of minerals to train/research the type.
Type	`Integer`
Range	[0–∞]
<Gas>	The required amount of gas to train/research the type.
Type	`Integer`
Range	[0–∞]
<SupplyOrEnergy>	For units: the required (or supplied, represented with a negative number) amount of supply (x2 compared to what is visible in the game). For upgrades (if applicable): the required amount of energy to use (as an ability).
Type	`Integer`
Range	[0–∞]
<BuildFrames>	The number of game frames that are required to complete training/researching the type.
Type	`Integer`
Range	[0–∞]
<RequiredUnitsOrUpgrades>	A (possibly empty) list of unit and/or upgrade types that are required to be present when starting to train/research the type. For upgrade types this is list is always either empty (for abilities that do not need to be research first) or of size one (indicating at which building type the upgrade needs to be researched).
Type	`List of Strings`
Range	See Sections 4.2 and 4.3

4.1.4 stats/6

Description Information about the static properties of unit types.

Syntax `stats(<Name>,<MaxHealth>,<MaxShield>,<MaxEnergy>, <TopSpeed>,<Conditions>)`

Example `stats('Protoss High Templar', 40, 40, 200, 32, [canMove,organic]).`

Parameters

<Name>	The name of the unit type.
Name	`String`
Range	See Section 4.2
<MaxHealth>	The maximum amount of health for units of the type. Invincible units (like spells) have 0 here.
Type	`Integer`
Range	[0–∞]
<MaxShield>	The maximum amount of shield for units of the type.
Type	`Integer`
Range	[0–∞]
<MaxEnergy>	The maximum amount of energy for units of the type. 0 for units that do not use energy at all.
Type	`Integer`
Range	[0–∞]
<TopSpeed>	The top movement speed for units of the type (without upgrades).
Type	`Integer`
Range	[0–∞]
<Conditions>	A (possibly empty) list static conditions for units of the type.
Type	`List of Strings`
Range	[addon, building, canBurrow, canDetect, canLift, canMove, canTrain, flies, mechanical, organic, requiresCreep, requiresPsi, robotic, spell]

4.1.5 metrics/5

Description: Information about the metrics (i.e. on the map) of unit types.

Syntax: `metrics(<Name>,<Width>,<Height>,<SightRange>,<SpaceRequired>)`

Example: `metrics('Terran Bunker', 3, 2, 10, -4).`

Parameters:

<Name>	The name of the unit type.
Name	`String`
Range	See Section 4.2
<Width>	The number of build tiles the unit is wide.
Type	`Integer`
Range	[0–∞]
<Height>	The number of build tile the unit is high.
Type	`Integer`
Range	[0–∞]
<SightRange>	The number of build tiles the unit's visibility reaches (without upgrades).
Type	`Integer`
Range	[0–∞]
<SpaceRequired>	The space required to load the unit into a loadable unit if positive; a negative number indicates the space a loadable unit provides for other units (although e.g. Overlords require an upgrade to actually use this space).
Type	`Integer`
Range	[-8–8]

4.1.6 combat/6

Description: Information about the offensive capabilities of unit types or upgrade types (i.e abilities used on units or locations). All damages take attack speed into account, but not specific trade-offs (e.g. damage against organic vs. mechanic units). None of the properties take possible upgrades into account.

Syntax:
```
combat(<Name>,<GroundDamage>,<AirDamage>,
  <CooldownFrames>,<Range>,<SplashRadius>)
```

Example:
```
combat('Psionic Storm', 14, 14, 45, 9, 1).
```

Parameters:

<Name>	The name of the unit or upgrade type (that can be used as an ability)
Name	`String`
Range	See Sections 4.2 and 4.3
<GroundDamage>	The amount of damage the type does to ground units. 0 means it cannot attack ground.
Type	`Integer`
Range	[0–∞]
<AirDamage>	The amount of damage the type does to air units. 0 means it cannot attack air.
Type	`Integer`
Range	[0–∞]
<CooldownFrames>	The number number of game frames the type needs to cool down after an attack (i.e. wait before launching another attack).
Type	`Integer`
Range	[0–∞]
<Range>	The number of build tiles the type's attack can span.
Type	`Integer`
Range	[0–∞]
<SplashRadius>	The number of build tiles the type's attack can do splash damage (0 means no splash).
Type	`Integer`
Range	[0-∞]

4.2 Unit Types

StarCraft's unit types (per race and category).

4.2.1 Terran

Ground

`Terran Firebat`
`Terran Ghost`
`Terran Goliath`
`Terran Marine`
`Terran Medic`
`Terran SCV`
`Terran Siege Tank`
`Terran Vulture`
`Terran Vulture Spider Mine`

Air

`Terran Battlecruiser`
`Terran Dropship`
`Terran Nuclear Missile`
`Terran Science Vessel`
`Terran Valkyrie`
`Terran Wraith`

Buildings

`Terran Academy`
`Terran Armory`
`Terran Barracks`
`Terran Bunker`
`Terran Command Center`
`Terran Engineering Bay`
`Terran Factory`
`Terran Missile Turret`
`Terran Refinery`
`Terran Science Facility`
`Terran Starport`
`Terran Supply Depot`

Addons
Terran Comsat Station
Terran Control Tower
Terran Covert Ops
Terran Machine Shop
Terran Nuclear Silo
Terran Physics Lab

Spells
Spell Scanner Sweep

4.2.2 Protoss

Ground
Protoss Archon
Protoss Dark Archon
Protoss Dark Templar
Protoss Dragoon
Protoss High Templar
Protoss Probe
Protoss Reaver
Protoss Scarab
Protoss Zealot

Air
Protoss Arbiter
Protoss Carrier
Protoss Corsair
Protoss Interceptor
Protoss Observer
Protoss Scout
Protoss Shuttle

Buildings
Protoss Arbiter Tribunal
Protoss Assimilator
Protoss Citadel of Adun
Protoss Cybernetics Core
Protoss Fleet Beacon
Protoss Forge

Protoss Gateway
Protoss Nexus
Protoss Observatory
Protoss Photon Cannon
Protoss Pylon
Protoss Robotics Facility
Protoss Robotics Support Bay
Protoss Shield Battery
Protoss Stargate
Protoss Templar Archives

Spells
Spell Disruption Web

4.2.3 Zerg

Ground
Zerg Broodling
Zerg Defiler
Zerg Drone
Zerg Egg
Zerg Hydralisk
Zerg Larva
Zerg Lurker
Zerg Lurker Egg
Zerg Ultralisk
Zerg Zergling

Air
Zerg Cocoon
Zerg Devourer
Zerg Guardian
Zerg Mutalisk
Zerg Overlord
Zerg Queen
Zerg Scourge

Buildings
Zerg Creep Colony
Zerg Defiler Mound

Zerg Evolution Chamber
Zerg Extractor
Zerg Greater Spire
Zerg Hatchery
Zerg Hive
Zerg Hydralisk Den
Zerg Lair
Zerg Nydus Canal
Zerg Queens Nest
Zerg Spawning Pool
Zerg Spire
Zerg Spore Colony
Zerg Sunken Colony
Zerg Ultralisk Cavern

Spells

Spell Dark Swarm

4.3 Upgrade Types

All upgrade types that can be researched for each race at the indicated building. Brackets with numbers indicate that the upgrade is available at multiple levels (i.e. *Zerg Carapace 1 > Zerg Carapace 2 > Zerg Carapace 3*). Upgrades that can be used as abilities are indicated in the next section.

4.3.1 Terran

Academy
`Caduceus Reactor`
`Optical Flare`
`Restoration`
`Stim Packs`
`U-238 Shells`

Armory
`Terran Vehicle Plating (1,2,3)`
`Terran Vehicle Weapons (1,2,3)`
`Terran Ship Plating (1,2,3)`
`Terran Ship Weapons (1,2,3)`

Control Tower
`Cloaking Field`

Covert Ops
`Lockdown`
`Moebius Reactor`
`Ocular Implants`
`Personnel Cloaking`

Engineering Bay
`Terran Infantry Armor (1,2,3)`
`Terran Infantry Weapons (1,2,3)`

Machine Shop
`Charon Boosters`
`Ion Thrusters`
`Spider Mines`
`Tank Siege Mode`

Physics Lab
Colossus Reactor
Yamato Gun

Science Facility
EMP Shockwave
Irradiate
Titan Reactor

Control Tower
Apollo Reactor

4.3.2 Protoss

Arbiter Tribunal
Khaydarin Core
Recall
Statis Field

Citadel of Adun
Leg Enhancements

Cybernetics Core
Protoss Air Armor (1,2,3)
Protoss Air Weapons (1,2,3)
Singularity Charge

Fleet Beacon
Apial Sensors
Argus Jewel
Carrier Capacity
Disruption Web
Gravitic Thrusters

Forge
Protoss Ground Armor (1,2,3)
Protoss Ground Weapons (1,2,3)
Protoss Plasma Shields (1,2,3)

Observatory
Gravitic Boosters
Sensor Array

Robotics Support Bay
Gravitic Drive
Reaver Capacity
Scarab Damage

Templar Archives
Argus Talisman
Hallucination
Khaydarin Amulet
Maelstrom
Mind Control
Psionic Storm

4.3.3 Zerg

Defiler Mound
Consume
Metasynaptic Node
Plague

Evolution Chamber
Zerg Carapace (1,2,3)
Zerg Melee Attacks (1,2,3)
Zerg Missile Attacks (1,2,3)

Hatchery
Burrowing

Hydralisk Den
Grooved Spines
Lurker Aspect
Muscular Augments

Lair / Hive
Antennae
Pneumatized Carapace
Ventral Sacs

Queens Nest
Ensnare
Gamete Meiosis
Spawn Broodlings

Spawning Pool
Adrenal Glands
Metabolic Boost

(Greater) Spire
Zerg Flyer Attacks (1,2,3)
Zerg Flyer Carapace (1,2,3)

Ultralisk Cavern
Anabolic Synthesis
Chitinous Plating

4.4 Abilities

All abilities that can be used by the indicated units of each race, usually after having the researched the corresponding upgrade (exceptions to this are noted). The potential target(s) is/are indicated in brackets.

4.4.1 Terran

Battle Cruiser
`Yamato Gun` (*unit*)

Comsat Station
`Scanner Sweep` (*position or unit*)

Ghost
`Lockdown` (*unit*)
`Personnel Cloaking` (*self*)
`Nuclear Strike` (*position or unit*)

Marine / Firebat
`Stim Packs` (*self*)

Medic
`Healing` (*position or unit*)
`Restoration` (*unit*)
`Optical Flare` (*unit*)

Science Vessel
`Defensive Matrix` (*unit*)
`EMP Shockwave` (*position or unit*)
`Irradiate` (*unit*)

Siege Tank
`Tank Siege Mode` (*self*)

Vulture
`Spider Mines` (*position*)

Wraith
`Cloaking Field` (*self*)

4.4.2 Protoss

Arbiter
`Recall` (*position or unit*)
`Stasis Field` (*position or unit*)

Corsair
`Disruption Web` (*position or unit*)

Dark Archon
`Feedback` (*unit*)
`Maelstrom` (*position or unit*)
`Mind Control` (*unit*)

Dark Templar
`Dark Archon Meld` (*unit*)

High Templar
`Archon Warp` (*unit*)
`Psionic Storm` (*position or unit*)
`Hallucination` (*unit*)

4.4.3 Zerg

Generic
`Burrowing` (*self*)
Lurkers can use this ability without having it researched; for Zerglings, Defilers, Hydralisks and Drones the upgrade needs to be researched.

Defiler
`Dark Swarm` (*position or unit*)
`Plague` (*position or unit*)
`Consume` (*unit*)

Queen
`Infestation` (*unit*)
`Parasite` (*unit*)

`Ensnare` (*position or unit*)
`Spawn Broodlings` (*unit*)

4.5 Known Limitations

Here we list functionalities that the connector does not support at this time:

- Getting more detailed information about the map like the exact shapes of regions and non-walkable locations.
- Detecting 'bullets' like Psionic Storms and Subterranean Spikes.
- Detecting upgrades performed by the enemy player.
- Specifying the random seed used for a match.
- Automatically pausing the game when a whole MAS is paused.