CHAPTER 11

BUILDING A TRADING PLAN

TRADING STRATEGIES

FROM BEGINNER TO PROFESSIONAL

versopropfirm.com

Building a Trading Plan

A trading plan is your personal constitution as a trader. Without it, you're navigating without a compass in an ocean of infinite volatility. As the old military proverb says: "No plan survives contact with the enemy, but without a plan, you're already defeated."

Anatomy of a Professional Trading Plan Trading Plan Structure:

1. Philosophy and Objectives
2. Personal Risk Profile
3. Specific Strategies
4. Capital Management
5. Execution Protocols
6. Performance Metrics
7. Review and Adaptation

Section 1: Philosophy and Objectives
python

```python
class TradingPlanFoundation:
    def __init__(self):
        self.trading_philosophy = self.define_philosophy()
        self.financial_objectives = self.set_objectives()
        self.time_commitment = self.assess_time_availability()

    def define_philosophy(self):
        """
        Define your fundamental trading philosophy
        """
        return {
            'market_view': {
                'question': 'How do you view the market?',
                'options': [
                    'Random walk - unpredictable movements',
                    'Pattern-based - repetitive patterns',
                    'Efficient but exploitable - temporary inefficiencies',
                    'Behavioral driven - mass psychology'
                ],
                'your_choice': None,  # Trader must select
                'implications': 'Determines which strategies to use'
            },
```

```
        'trading_style_preference': {
            'question': 'What is your natural style?',
            'assessment': {
                'patience_level': 'Scale 1-10',
                'stress_tolerance': 'Scale 1-10',
                'analytical_vs_intuitive': 'Scale 1-10 (1=pure analytical)',
                'available_screen_time': 'Hours per day'
            }
        },
        'risk_philosophy': {
            'question': 'How do you view risk?',
            'perspectives': [
                'Risk to be minimized at all costs',
                'Risk as necessary cost of returns',
                'Risk as opportunity when managed properly',
                'Calculated risk-taking for asymmetric returns'
            ]
        }
    }

def set_objectives(self):
    """
    Define specific SMART objectives
    """
    return {
        'financial_targets': {
            'annual_return_target': {
                'conservative': '8-12%',
                'moderate': '15-25%',
                'aggressive': '30%+',
                'your_target': None  # Trader specifies
            },
            'maximum_acceptable_drawdown': {
                'conservative': '5-8%',
                'moderate': '12-18%',
                'aggressive': '25%+',
                'your_limit': None
            },
            'capital_growth_timeline': {
                '1_year': 'Target %',
                '3_years': 'Target %',
                '5_years': 'Target %'
```

```
            }
        },
        'skill_development_goals': [
            'Master specific strategy',
            'Improve win rate by X%',
            'Develop new market expertise',
            'Enhance risk management'
        ],
        'lifestyle_objectives': {
            'income_replacement': 'Replace X% of salary',
            'time_freedom': 'Reduce trading hours while maintaining returns',
            'stress_reduction': 'Maintain profitability with less stress'
        }
    }
```

Section 2: Personal Risk Profile

python

```
class PersonalRiskProfile:
    def __init__(self):
        self.risk_assessment = self.comprehensive_risk_assessment()
        self.capital_allocation = self.determine_capital_allocation()

    def comprehensive_risk_assessment(self):
        """
        Comprehensive risk profile assessment
        """
        return {
            'financial_situation': {
                'net_worth': 'Total assets minus liabilities',
                'liquid_assets': 'Cash and easily convertible assets',
                'trading_capital': 'Amount specifically for trading',
                'emergency_fund': 'Months of expenses covered',
                'debt_obligations': 'Monthly debt payments',
                'income_stability': 'Consistency of primary income'
            },
            'risk_tolerance_questionnaire': {
                'questions': [
                    {
                        'q': 'If your portfolio lost 20% in a month, would you:',
                        'options': {
                            'a': 'Sell everything immediately',
                            'b': 'Reduce position sizes significantly',
```

```
                    'c': 'Hold steady and analyze',
                    'd': 'Buy more at lower prices'
                },
                'scoring': {'a': 1, 'b': 2, 'c': 3, 'd': 4}
            },
            {
                'q': 'Your trading capital represents what % of net worth?',
                'options': {
                    'a': 'More than 50%',
                    'b': '25-50%',
                    'c': '10-25%',
                    'd': 'Less than 10%'
                },
                'scoring': {'a': 1, 'b': 2, 'c': 3, 'd': 4}
            }
        ]
    },
    'behavioral_tendencies': {
        'loss_aversion': 'How much do losses hurt vs gains feel good?',
        'overconfidence': 'Tendency to overestimate abilities',
        'herding': 'Following crowd vs independent thinking',
        'anchoring': 'Difficulty adjusting from first impression'
    }
}

def calculate_optimal_position_sizing(self, account_size, risk_tolerance_score):
    """
    Calculates optimal position sizing based on risk profile
    """
    # Base risk per trade based on tolerance score
    base_risk_mapping = {
        1: 0.005,  # Ultra conservative - 0.5%
        2: 0.01,   # Conservative - 1%
        3: 0.02,   # Moderate - 2%
        4: 0.03,   # Aggressive - 3%
        5: 0.05    # Very Aggressive - 5%
    }
    base_risk = base_risk_mapping.get(risk_tolerance_score, 0.02)
    # Adjustments based on financial situation
    if account_size < 25000:  # Small account
        base_risk *= 0.5  # Reduce risk
    elif account_size > 500000:  # Large account
```

```
            base_risk *= 1.2  # Can take slightly more risk
        return {
            'base_risk_per_trade': base_risk,
            'max_portfolio_risk': base_risk * 3,  # Max 3 positions at full size
            'correlation_adjustment': 0.7,  # Reduce size for correlated positions
            'volatility_adjustment': {
                'low_vol': 1.2,
                'normal_vol': 1.0,
                'high_vol': 0.8
            }
        }
```

Section 3: Documented Specific Strategies

python

```
class StrategyDocumentation:
    def __init__(self):
        self.strategies = {}

    def document_strategy(self, strategy_name):
        """
        Complete template for documenting each strategy
        """
        return {
            'strategy_name': strategy_name,
            'market_conditions': {
                'optimal_conditions': 'When it works best',
                'avoid_conditions': 'When not to use',
                'volatility_requirements': 'Minimum/maximum ATR',
                'volume_requirements': 'Minimum daily volume',
                'time_of_day': 'Optimal trading hours'
            },
            'entry_criteria': {
                'primary_setup': 'Main setup',
                'confirmation_signals': 'Required confirmation signals',
                'filters': 'Additional filters',
                'disqualifiers': 'What invalidates the setup'
            },
            'position_management': {
                'initial_position_size': 'Initial size calculation',
                'stop_loss_method': 'How to determine stop loss',
                'profit_targets': 'Profit-taking levels',
                'scaling_rules': 'When to add/reduce position',
```

```
            'time_stops': 'Time limits in position'
        },
        'backtesting_results': {
            'test_period': 'Test period',
            'total_trades': 'Number of trades',
            'win_rate': 'Winning percentage',
            'avg_win_loss': 'Average win/loss ratio',
            'max_drawdown': 'Maximum consecutive loss',
            'sharpe_ratio': 'Risk-adjusted return'
        },
        'execution_checklist': [
            'Market analysis complete?',
            'Setup meets all criteria?',
            'Risk calculated properly?',
            'Stop loss placed?',
            'Profit targets set?',
            'Position size appropriate?'
        ]
    }

def create_strategy_scorecard(self, strategy_name, market_data):
    """
    Real-time scorecard to evaluate setup
    """
    return {
        'strategy': strategy_name,
        'timestamp': datetime.now(),
        'setup_quality': {
            'primary_criteria_met': self.check_primary_criteria(market_data),
            'confirmation_strength': self.rate_confirmation(market_data),
            'risk_reward_ratio': self.calculate_risk_reward(market_data),
            'market_conditions_score': self.assess_conditions(market_data)
        },
        'overall_score': 0,  # Calculated from components
        'recommendation': 'PASS/CONSIDER/STRONG_BUY',
        'risk_adjustment': 'Factor to adjust normal position size'
    }
```

Section 4: Capital Management Protocols

python

```
class CapitalManagementProtocol:
    def __init__(self, total_capital):
```

```
        self.total_capital = total_capital
        self.allocation_rules = self.define_allocation_rules()

    def define_allocation_rules(self):
        """
        Defines strict capital allocation rules
        """
        return {
            'core_allocation': {
                'trading_capital': 0.70,  # 70% for active trading
                'cash_reserve': 0.20,     # 20% cash for opportunities
                'emergency_fund': 0.10    # 10% never touch
            },
            'strategy_allocation': {
                'primary_strategy': 0.40,  # 40% in primary strategy
                'secondary_strategy': 0.25,  # 25% in secondary strategy
                'experimental': 0.05       # 5% for testing new ideas
            },
            'correlation_limits': {
                'max_single_sector': 0.25,  # Maximum 25% in one sector
                'max_correlated_pairs': 0.15,  # Maximum 15% in correlated pairs
                'geographic_diversification': 0.30  # Maximum 30% in one region
            },
            'drawdown_protocols': {
                'level_1': {
                    'trigger': '5% drawdown',
                    'action': 'Reduce all position sizes by 25%'
                },
                'level_2': {
                    'trigger': '10% drawdown',
                    'action': 'Reduce position sizes by 50%, review all strategies'
                },
                'level_3': {
                    'trigger': '15% drawdown',
                    'action': 'Stop trading, mandatory cooling-off period'
                }
            }
        }

    def daily_capital_check(self, current_portfolio_value):
        """
        Daily capital status check
```

```
        """
        total_return = (current_portfolio_value / self.total_capital) - 1
        current_drawdown = self.calculate_current_drawdown(current_portfolio_value)
        status = {
            'current_value': current_portfolio_value,
            'total_return': total_return,
            'current_drawdown': current_drawdown,
            'allocation_status': self.check_allocation_compliance(),
            'risk_budget_used': self.calculate_risk_budget_usage(),
            'action_required': self.determine_required_actions(current_drawdown)
        }
        return status

    def position_sizing_calculator(self, trade_setup, current_market_volatility):
        """
        Dynamic position sizing calculator
        """
        base_position_size = self.total_capital * 0.02  # 2% base risk
        # Adjustments based on setup quality
        quality_multiplier = {
            'A+': 1.5,  # Exceptional setup
            'A': 1.2,   # Very good setup
            'B': 1.0,   # Standard setup
            'C': 0.7,   # Marginal setup
            'D': 0.0    # Do not trade
        }[trade_setup['quality_grade']]
        # Adjustment for market volatility
        if current_market_volatility > 1.5 * trade_setup['normal_volatility']:
            volatility_adjustment = 0.7
        elif current_market_volatility < 0.7 * trade_setup['normal_volatility']:
            volatility_adjustment = 1.2
        else:
            volatility_adjustment = 1.0
        # Adjustment for correlation with existing positions
        correlation_adjustment =
self.calculate_correlation_adjustment(trade_setup['symbol'])
        final_position_size = (base_position_size * quality_multiplier *
volatility_adjustment * correlation_adjustment)
        return {
            'recommended_size': final_position_size,
            'max_loss_amount': final_position_size * trade_setup['stop_distance'],
            'adjustments_applied': {
```

```
            'quality': quality_multiplier,
            'volatility': volatility_adjustment,
            'correlation': correlation_adjustment
        }
    }
```

Section 5: Execution Protocols

python

```
class ExecutionProtocols:
    def __init__(self):
        self.pre_trade_checklist = self.create_pre_trade_checklist()
        self.execution_rules = self.define_execution_rules()
        self.post_trade_procedures = self.create_post_trade_procedures()

    def create_pre_trade_checklist(self):
        """
        Mandatory checklist before each trade
        """
        return [
            {
                'category': 'Market Analysis',
                'items': [
                    'Major trend identified and confirmed',
                    'Key support/resistance levels marked',
                    'Volume analysis completed',
                    'News/events checked for next 24 hours'
                ]
            },
            {
                'category': 'Setup Validation',
                'items': [
                    'Primary setup criteria 100% met',
                    'Minimum 2 confirmation signals present',
                    'No disqualifying factors present',
                    'Risk/reward ratio minimum 1:2'
                ]
            },
            {
                'category': 'Risk Management',
                'items': [
                    'Position size calculated and appropriate',
                    'Stop loss level determined and logical',
```

```
                'Portfolio exposure within limits',
                'Correlation with existing positions checked'
            ]
        },
        {
            'category': 'Execution Readiness',
            'items': [
                'Internet connection stable',
                'Platform functioning properly',
                'Sufficient buying power available',
                'Mental state appropriate for trading'
            ]
        }
    ]

def execution_decision_tree(self, setup_data):
    """
    Decision tree for trade execution
    """
    decision_tree = {
        'step_1': {
            'question': 'Does setup meet ALL primary criteria?',
            'yes': 'step_2',
            'no': 'REJECT_TRADE'
        },
        'step_2': {
            'question': 'Is risk/reward ratio >= 2:1?',
            'yes': 'step_3',
            'no': 'REDUCE_POSITION_OR_REJECT'
        },
        'step_3': {
            'question': 'Is market volatility within acceptable range?',
            'yes': 'step_4',
            'no': 'ADJUST_POSITION_SIZE'
        },
        'step_4': {
            'question': 'Is portfolio correlation within limits?',
            'yes': 'step_5',
            'no': 'REDUCE_SIZE_OR_WAIT'
        },
        'step_5': {
            'question': 'Is mental/emotional state optimal?',
```

```
            'yes': 'EXECUTE_TRADE',
            'no': 'WAIT_OR_REDUCE_SIZE'
        }
    }
    return self.process_decision_tree(decision_tree, setup_data)

def order_management_protocol(self, trade_details):
    """
    Specific protocol for order management
    """
    return {
        'entry_orders': {
            'market_hours_entry': {
                'order_type': 'Market order',
                'timing': 'Immediate execution during liquid hours',
                'slippage_limit': '0.05% of entry price'
            },
            'pre_market_entry': {
                'order_type': 'Limit order',
                'limit_price': 'Entry signal price + 2 ticks',
                'time_in_force': 'Day order'
            },
            'breakout_entry': {
                'order_type': 'Stop-limit order',
                'stop_price': 'Breakout level',
                'limit_price': 'Stop price + 0.1%'
            }
        },
        'stop_loss_orders': {
            'initial_stop': {
                'type': 'Stop-loss order',
                'placement': 'Immediately after entry fill',
                'location': 'Predetermined technical level'
            },
            'trailing_stop': {
                'activation': 'When profit >= 1R',
                'trail_amount': '50% of profit',
                'minimum_trail': 'Breakeven + commission'
            }
        },
        'profit_taking': {
            'partial_exits': [
```

```
                {'level': '1R profit', 'percentage': '50%'},
                {'level': '2R profit', 'percentage': '25%'},
                {'level': '3R+ profit', 'percentage': '25%'}
            ],
            'time_exits': {
                'intraday': 'Close 15 minutes before market close',
                'swing': 'Review after 5 trading days',
                'position': 'Review monthly'
            }
        }
    }
```

Section 6: Metrics and KPIs System

python

```
class PerformanceMetricsSystem:
    def __init__(self):
        self.kpi_definitions = self.define_key_metrics()
        self.benchmarks = self.establish_benchmarks()

    def define_key_metrics(self):
        """
        Defines key metrics with exact formulas
        """
        return {
            'profitability_metrics': {
                'total_return': {
                    'formula': '(Ending Value - Starting Value) / Starting Value',
                    'target': 'Varies by risk tolerance',
                    'frequency': 'Daily tracking, monthly evaluation'
                },
                'sharpe_ratio': {
                    'formula': '(Portfolio Return - Risk Free Rate) / Portfolio Std Dev',
                    'target': '> 1.0 (good), > 1.5 (excellent)',
                    'frequency': 'Monthly calculation'
                },
                'calmar_ratio': {
                    'formula': 'Annual Return / Maximum Drawdown',
                    'target': '> 2.0 (good), > 3.0 (excellent)',
                    'frequency': 'Quarterly calculation'
                }
            },
            'risk_metrics': {
```

```
            'maximum_drawdown': {
                'formula': 'Largest peak-to-trough decline',
                'target': '< 15% for moderate risk',
                'frequency': 'Daily monitoring'
            },
            'value_at_risk': {
                'formula': '95th percentile worst daily loss',
                'target': '< 2% of portfolio value',
                'frequency': 'Weekly calculation'
            },
            'beta': {
                'formula': 'Covariance(Portfolio, Market) / Variance(Market)',
                'target': 'Depends on strategy (0.5-1.5)',
                'frequency': 'Monthly calculation'
            }
        },
        'efficiency_metrics': {
            'win_rate': {
                'formula': 'Winning Trades / Total Trades',
                'target': '> 50% for most strategies',
                'frequency': 'Weekly review'
            },
            'profit_factor': {
                'formula': 'Gross Profit / Gross Loss',
                'target': '> 1.5 (good), > 2.0 (excellent)',
                'frequency': 'Monthly calculation'
            },
            'average_trade': {
                'formula': 'Total P&L / Number of Trades',
                'target': 'Positive and growing',
                'frequency': 'Weekly review'
            }
        }
    }

def automated_reporting_system(self, trading_data):
    """
    Automated reporting system
    """
    return {
        'daily_report': {
            'trades_executed': len(trading_data.get('daily_trades', [])),
```

```
            'daily_pnl': self.calculate_daily_pnl(trading_data),
            'positions_held': self.count_open_positions(trading_data),
            'risk_utilization': self.calculate_risk_usage(trading_data),
            'alerts': self.generate_daily_alerts(trading_data)
        },
        'weekly_report': {
            'weekly_return': self.calculate_weekly_return(trading_data),
            'strategy_performance': self.analyze_strategy_performance(trading_data),
            'risk_metrics_update': self.update_risk_metrics(trading_data),
            'goal_progress': self.track_goal_progress(trading_data)
        },
        'monthly_report': {
            'comprehensive_analysis': self.full_performance_analysis(trading_data),
            'benchmark_comparison': self.compare_to_benchmarks(trading_data),
            'strategy_optimization':
self.identify_optimization_opportunities(trading_data),
            'plan_adjustments': self.recommend_plan_changes(trading_data)
        }
    }

def performance_dashboard_config(self):
    """
    Configuration for real-time performance dashboard
    """
    return {
        'real_time_widgets': [
            {
                'widget': 'P&L Gauge',
                'data_source': 'current_unrealized_pnl',
                'update_frequency': '1 second',
                'alert_thresholds': [-2, -5, 5, 10]  # Percentage levels
            },
            {
                'widget': 'Risk Utilization Bar',
                'data_source': 'portfolio_risk_percentage',
                'update_frequency': '5 seconds',
                'max_threshold': 100
            },
            {
                'widget': 'Open Positions Table',
                'data_source': 'current_positions',
                'update_frequency': '10 seconds',
```

```
                'columns': ['Symbol', 'Size', 'Entry', 'Current', 'P&L', '%']
            }
        ],
        'historical_charts': [
            {
                'chart': 'Equity Curve',
                'timeframes': ['1D', '1W', '1M', '3M', '1Y', 'ALL'],
                'overlays': ['Benchmark', 'Drawdown'],
                'annotations': ['Major trades', 'Strategy changes']
            },
            {
                'chart': 'Rolling Sharpe Ratio',
                'timeframe': '30-day rolling',
                'benchmark_line': 1.0,
                'target_zone': [1.0, 2.0]
            }
        ]
    }
```

Section 7: Review and Continuous Improvement Process

python

```
class ContinuousImprovementProcess:
    def __init__(self):
        self.review_schedule = self.establish_review_schedule()
        self.improvement_framework = self.create_improvement_framework()

    def establish_review_schedule(self):
        """
        Structured review schedule
        """
        return {
            'daily_reviews': {
                'timing': 'End of trading day',
                'duration': '15-20 minutes',
                'focus_areas': [
                    'Trades executed today',
                    'Adherence to plan',
                    'Emotional state throughout day',
                    'Market observations',
                    'Tomorrow\'s preparation'
                ]
            },
```

```
            'weekly_reviews': {
                'timing': 'Weekend',
                'duration': '1-2 hours',
                'focus_areas': [
                    'Weekly performance vs goals',
                    'Strategy effectiveness analysis',
                    'Risk management assessment',
                    'Pattern identification in trading',
                    'Market regime analysis'
                ]
            },
            'monthly_reviews': {
                'timing': 'First weekend of new month',
                'duration': '3-4 hours',
                'focus_areas': [
                    'Comprehensive performance analysis',
                    'Goal progress evaluation',
                    'Strategy parameter optimization',
                    'Plan adjustments needed',
                    'Skills development planning'
                ]
            },
            'quarterly_reviews': {
                'timing': 'End of each quarter',
                'duration': 'Full day',
                'focus_areas': [
                    'Complete strategy overhaul',
                    'Market cycle analysis',
                    'Capital allocation review',
                    'Technology and tools upgrade',
                    'Education and development plan'
                ]
            }
        }

    def create_improvement_framework(self):
        """
        Systematic framework for continuous improvement
        """
        return {
            'identification_phase': {
                'data_collection': [
```

```
            'Trading journal analysis',
            'Performance metrics review',
            'Market condition correlation',
            'Emotional pattern identification'
        ],
        'problem_identification': [
            'Underperforming strategies',
            'Recurring mistakes',
            'Risk management failures',
            'Execution issues'
        ]
    },
    'analysis_phase': {
        'root_cause_analysis': [
            'Why did this pattern occur?',
            'What are the contributing factors?',
            'Is it systematic or random?',
            'What market conditions triggered it?'
        ],
        'impact_assessment': [
            'How much did it cost?',
            'How frequently does it occur?',
            'What\'s the opportunity cost?',
            'How does it affect other areas?'
        ]
    },
    'solution_phase': {
        'solution_brainstorming': [
            'Multiple solution approaches',
            'Cost-benefit analysis',
            'Implementation difficulty',
            'Risk of unintended consequences'
        ],
        'testing_methodology': [
            'Paper trading validation',
            'Small position testing',
            'Limited time testing',
            'Performance measurement'
        ]
    },
    'implementation_phase': {
        'gradual_rollout': [
```

```
                'Start with smallest impact area',
                'Monitor results closely',
                'Adjust based on feedback',
                'Scale up if successful'
            ],
            'documentation': [
                'Record all changes made',
                'Document reasoning',
                'Set success metrics',
                'Plan review schedule'
            ]
        }
    }

def plan_evolution_tracker(self):
    """
    System to track trading plan evolution
    """
    return {
        'version_control': {
            'current_version': '1.0',
            'version_history': [],
            'change_log': [
                {
                    'version': '1.1',
                    'date': '2024-02-15',
                    'changes': 'Adjusted position sizing after drawdown analysis',
                    'reason': 'Risk management improvement',
                    'impact': 'Reduced max position size by 25%'
                }
            ]
        },
        'adaptation_triggers': [
            'Significant change in market regime',
            'Personal financial situation change',
            'Consistent underperformance vs benchmarks',
            'New strategy development',
            'Risk tolerance evolution'
        ],
        'success_metrics_evolution': {
            'initial_targets': 'Original goals set',
            'adjusted_targets': 'Revised based on experience',
```

```
            'achievement_tracking': 'Progress toward goals',
            'target_reasonableness': 'Are targets realistic?'
        }
    }
```

Final Trading Plan Template

python

```
def generate_complete_trading_plan(trader_profile):
    """
    Generates a complete personalized trading plan
    """
    trading_plan = {
        'header': {
            'trader_name': trader_profile['name'],
            'plan_version': '1.0',
            'creation_date': datetime.now().strftime('%Y-%m-%d'),
            'next_review_date': (datetime.now() + timedelta(days=30)).strftime('%Y-%m-
%d')
        },
        'executive_summary': {
            'trading_philosophy': trader_profile['philosophy'],
            'primary_objectives': trader_profile['objectives'],
            'risk_tolerance': trader_profile['risk_level'],
            'expected_annual_return': trader_profile['return_target'],
            'maximum_drawdown_limit': trader_profile['drawdown_limit']
        },
        'strategy_specifications': trader_profile['strategies'],
        'risk_management_rules': trader_profile['risk_rules'],
        'execution_protocols': trader_profile['execution_rules'],
        'performance_measurement': trader_profile['kpis'],
        'review_and_improvement': {
            'review_schedule': trader_profile['review_schedule'],
            'success_criteria': trader_profile['success_metrics'],
            'failure_protocols': trader_profile['failure_responses']
        }
    }
    return trading_plan
```

versopropfirm.com